VAX/VMS
Introduction to VAX/VMS

Order Number: AA-Y500A-TE

September 1984

This book introduces the novice user to VAX/VMS. It discusses the DIGITAL Command Language (DCL), the Mail and Phone utilities, file manipulation, program development, and basic system concepts.

Revision/Update Information: This is a new manual.

Software Version: VAX/VMS Version 4.0

digital equipment corporation
maynard, massachusetts

September, 1984

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	RSX
DEC/CMS	EduSystem	UNIBUS
DEC/MMS	IAS	VAX
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter		

digital

ZK2315

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a registered trademark of the American Mathematical Society.

Contents

TABLES

1 Getting Started

1.1 Things to Remember

The following chart explains the graphic conventions used in this manual.

What You See	What it Means
RET DEL TAB	These symbols indicate that you press the RETURN, DELETE, or TAB key on the terminal.
CTRL/Y CTRL/Z	These symbols indicate that you hold down the CTRL key while you press a keyboard key, for example, Y. These symbols represent **control key sequences**. In some examples, control key sequences are shown as a circumflex (^) and a letter, for example ^Y, because that is how they are displayed by the system.
All **commands** are	A word or phrase in **bold** indicates a term defined in the GLOSSARY at the end of this guide.
$ SHOW TIME 4-FEB-1985 10:25:43	**Interactive** examples, in which you enter a command and the system responds, are displayed as the following example shows: all the lines you type are shown in red letters, and everything the system prints or displays is shown in black letters.

To communicate with the VAX/VMS **operating system** you use a **terminal** that is connected to the computer. You tell the operating system what to do by typing a **command** on the **keyboard**. The system responds by executing your command.

If the system cannot interpret what you type, it displays an error message at your terminal; when the command has been successfully executed, you can type another command.

When you communicate with the system in this manner, you are an **interactive** user. A **batch** user, in contrast, communicates with the system by submitting all commands at one time in a batch **job**. This manual emphasizes how to use VAX/VMS interactively.

1.2 Looking at Terminals and Keyboards

This section introduces several terminals and explains how to use your keyboard to communicate with the system.

1.2.1 Terminals

The terminals you can use to communicate with the VAX/VMS operating system fall into two general categories: hardcopy terminals and video display terminals.

Hardcopy terminals print on continuous forms of paper. Figure 1–1 shows one type of hardcopy terminal, the LA120.

Figure 1–1 The LA120 Terminal

Video display terminals display your input and system responses on a screen similar to that of a television. Figure 1–2 shows an example of a video display terminal, the VT100.

1.2.2 Keyboards

Figure 1–3 shows the keyboard layouts of the LA120, VT100, and VT200 Series terminals. All terminal keyboards have the same basic configuration as typewriter keyboards. However, a terminal has additional keys that send special signals to the operating system; to use this keyboard effectively, you should become familiar with these keys.

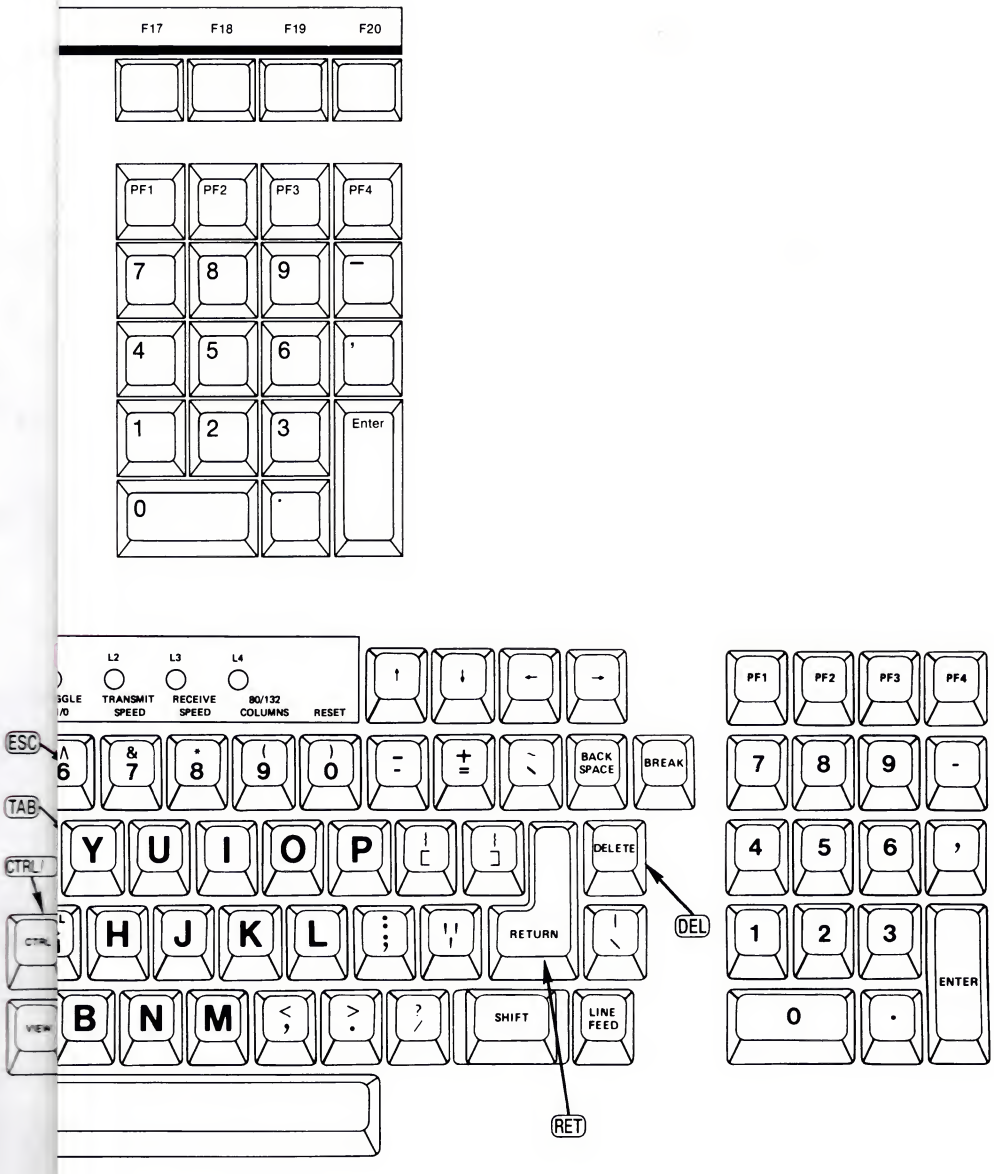
Figure 1–2 The VT100 Terminal

1.3 Logging In

Before you can log in to the system you must have an **account**. Accounts are set up by the system manager or whoever is responsible at your installation for authorizing the use of the system. This person will provide you with a **user name** and a **password**.

To begin a session at your terminal, you must **log in**. Logging in consists of getting the system's attention and identifying yourself as an authorized user.

Figu



VT100

ZK-1939-84

Your user name is a unique name that identifies you to the system and distinguishes you from other users. In many cases, a user name is your first or last name.

Your password, however, is for your protection. If you maintain its secrecy, other users cannot use system resources under your user name.

When you log in, you must enter both your user name and your password before VAX/VMS allows you to access the system.

1.3.1 Getting the Terminal Ready

Before you begin to use your terminal, be sure that:

- The terminal is plugged in and the power is turned on.
- If the terminal has a LOCAL/REMOTE switch, the switch is set to REMOTE.

The terminal should then be ready to accept your login. If you have any problems with the login procedure described in the next section, get help from the system **operator** or **system manager**. The terminal may not be properly connected to the computer, or the baud rate (the speed at which the terminal transmits or receives characters) may not be correctly set. It is possible that your system manager may tell you to press the BREAK key to begin the login procedure.

1.3.2 Gaining Access to the System

Press RETURN to signal the system that you want to log in. (You may need to press RETURN several times.) The system responds by prompting you for your user name. Enter your user name and press RETURN. The system displays your user name on the screen as you type it. After you enter your user name and press RETURN, the system prompts you for your password. When you type the password, the system does not display it; this preserves the secrecy of your password.

You are now at the DCL command level and ready to start typing commands.

The following example shows how to log in:

```
RET
Username: MAYMON RET
Password: RET
        Welcome to VAX/VMS version 4.0
  Last interactive login on Tuesday, 24-DEC-1984 08:41
  Last non-interactive login on Tuesday, 24-DEC-1984 11:05
$
```

The dollar sign is a symbol the system uses as a **prompt**. When VAX/VMS displays this character in the left margin, it indicates that the login was successful and that you can begin entering commands to the system.

Because VAX/VMS uses the dollar sign as the default system prompt, all the examples in this manual display the dollar sign as the prompt character. However, you should be aware that both you and your system manager have the ability to change the system prompt to some other character or string of characters. (Chapter 6 describes the SET PROMPT command.)

Note that if you type your user name or your password incorrectly, the system displays an error message. When this error message appears, you must repeat the login procedure.

1.4 Using the DIGITAL Command Language ("Telling VAX/VMS What to Do")

You can use the DIGITAL Command Language (DCL) to communicate with VAX/VMS. The DIGITAL Command Language is made up of DCL commands. These DCL commands are then read and translated by the **command interpreter**.

1.4.1 What are DCL Commands?

DCL commands are words, generally verbs, that describe the functions they perform. You can type them in upper or lower case. The following example demonstrates the DCL command SHOW:

```
$ SHOW TIME RET
```

The system responds to this command by displaying the current date and time, as follows:

```
  17-JUL-1985 11:55:40
$
```

The commands are part of the DIGITAL Command Language (DCL), which has its own vocabulary and rules of grammar. The vocabulary consists of commands, **parameters**, and **qualifiers**. The grammar consists of rules for using these terms.

Command parameters define what the command will act upon, and command qualifiers further define how that action will occur. For instance, the following PRINT command requires an object, or parameter, to indicate what is to be printed:

```
$ PRINT myfile.lis RET
```

In this command, MYFILE.LIS is a parameter for the PRINT command, indicating the name of the **file** to be printed. A space always separates the command from its parameter.

You can use command qualifiers to restrict or modify the function the command is to perform. The following example shows how to use the /COPIES qualifier to specify that two copies be printed:

```
$ PRINT/COPIES=2 myfile.lis RET
    Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

The entire **command string** includes the command and any parameters or qualifiers it may have. The keywords (commands, qualifiers, and parameters) that make up the DIGITAL Command Language have predefined meanings; therefore, you must use them exactly as defined and in some cases supply a **value** to complete them. For example, the /COPIES qualifier for the PRINT command requires a value: you supply the number of copies you want printed.

The rules of grammar and **syntax** for the DIGITAL Command Language (that is, the order of the words, the spacing, and the punctuation) are also defined. The *VAX/VMS DCL Dictionary* contains a dictionary of DCL commands and discusses the rules of grammar and syntax.

1.4.2 Responding to Command Prompts

When you enter a command at the terminal, you do not need to enter the entire command on one line. If you enter a command without specifying required parameters, the system prompts you for the additional information it requires, as the following example shows:

```
$ PRINT RET
$_File: myfile.dat RET
```

If a command requires two or more parameters, it prompts you for each parameter. In response to each prompt, you can enter just the prompted parameter or all the remaining parameters. In the following example, each file name is entered separately:

```
$ COPY RET
$_From:file1.dat RET
$_To:file2.dat RET
```

You could, however, enter both file names after the first prompt, as the following example shows:

```
$ COPY RET
$_From:file1.dat file2.dat RET
```

You could also enter the entire command string on one line:

```
$ COPY file1.dat file2.dat RET
```

1.4.3 Correcting Typing Errors

Some of the keys on the keyboard in Figure 1–3 provide editing functions that you can use to correct mistakes you make while typing commands. The following table describes these keys:

DELETE

Moves the cursor back by character (on the current line), then deletes that character. Most video display terminals actually move the **cursor** (an underline or block that marks your position) backward and erase the character when you press DELETE. Hardcopy terminals print a backslash character \, then each deleted character, then another backslash before printing the next character you enter. On some terminals, the key that performs the delete function is labeled RUBOUT.

CTRL/B (or Up Arrow)

Displays previously entered commands so that you can reprocess them. Everytime you press CTRL/B (or the Up Arrow), the previous command is displayed. (CTRL/B remembers up to 20 previously entered commands.) Press the Down Arrow to display commands in the other direction. You can edit the command string by pressing the left and right arrow keys, which move the cursor. You can also move the cursor to the beginning of the line by pressing CTRL/H and to the end of the line by pressing CTRL/E. Then, you can overstrike the character you want to change. Or, you can press CTRL/A, which enables you to insert a character rather than overstrike it.

CTRL/U

Deletes the current line and performs a return so you can reenter the entire line. Use CTRL/U when a line contains a number of mistakes and it would be tedious to use the DELETE key.

CTRL/C and CTRL/Y

Cancel an entire command, regardless of how many lines were used to enter it. You can also use CTRL/Y or CTRL/C to interrupt the system while it is executing a command. Such an interruption is useful in cases when you have entered a command and you want to stop it. After you press CTRL/Y, the DCL prompt returns. In the following example, CTRL/Y interrupts the typing of a long file:

```
$ TYPE myfile.lis RET
    .
    .
    .
CTRL/Y
$
```

You can use CTRL/C to abort an operation while staying in a utility. For example, if you start to send a message in MAIL and change your mind, press CTRL/C. The message will not be sent and you will remain in MAIL.

CTRL/W

Refreshes the screen. You commonly use CTRL/W while working in an editor when your text display has been overwritten by a system message.

1.5 Recognizing System Responses

The system can respond to your command in several ways. It can execute the command. Generally, you know your command has executed successfully when the system prompt returns (by default, the dollar sign). It can execute the command and inform you in a message of what it has done. It can, if execution is not successful, inform you of errors. It can even act for you, supplying values (defaults) that you have not supplied yourself.

1.5.1 What are Defaults?

A **default** is the value supplied by the operating system when you do not specify one yourself. For instance, if you do not specify the number of copies as a qualifier for the PRINT command, the system uses the default value of 1. By not explicitly stating a value, the system assumes that you have chosen the default. VAX/VMS supplies default values in several areas, including command qualifiers and parameters. The defaults used with individual commands are specified with each command's description in the *VAX/VMS DCL Dictionary*.

1.5.2 Looking at Informational Messages

The system responds to some commands by giving you information about what it has done. For example, when you use the PRINT command, the system displays the job identification number it assigned to the print job and shows the name of the print **queue** the job has entered.

```
$ PRINT myfile.lis[RET]
    Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

Not all commands display informational messages; in fact, successful completion of a command is most commonly indicated by a dollar sign prompt for another command. Unsuccessful completion is always indicated by one or more error messages.

1.5.3 Looking at Error Messages

If you enter a command incorrectly, the system displays an error message and prompts you for a command string as if no command had been entered, as the following example shows:

```
$ CAPY RET
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \CAPY\
$
```

The three-part code preceding the text of the message indicates that the message is:

- From DCL, the default command interpreter
- A warning (W) message
- Can be identified by the mnemonic IVVERB

You can also receive error messages during command execution if the system cannot perform the function you have requested. For example, if you type a PRINT command correctly, but the file that you specify does not exist, the PRINT command informs you of the error with a message like the following:

```
$ PRINT nofile.dat RET
%PRINT-E-OPENIN, error opening CLASS1:[MAYMON]NOFILE.DAT; as input
-RMS-E-FNF, file not found
%PRINT-F-CREJOB, error creating job
-JBC-E-EMPTYJOB, no files in job
$
```

The first message is from the PRINT command: it tells you it cannot open the specified file. The second message indicates the reason for the first, that is, the file cannot be found. "RMS" refers to the VAX/VMS file handling facility, Record Management Services; error messages related to file handling are generally VAX RMS messages.

1.5.4 Is the System Still Alive?

If you suspect that your system is not doing what you think it should be doing, press CTRL/T. CTRL/T displays a single line of statistical information about the current process. When you press CTRL/T during an interactive terminal session, it momentarily interrupts the current command, command procedure, or image in order to display statistics. The statistical information includes the node and user names, the current time, the current process, CPU usage, number of page faults, level of I/O activity, and memory usage. The following example shows a user named DAVIS on node YORK using the EDT editor:

```
YORK::DAVIS  13:45:02 EDT        CPU=00:00:03.33 PF=778 IO=295 MEM=315
```

If you know that your system is running and CTRL/T does not display statistical information, enter the SET CONTROL=T command and try again.

1.6 Using the HELP Command

When you use the VAX/VMS operating system, you may not always have a reference manual available at your terminal, and you may want to see the format of a command before you enter it. The HELP command is designed to provide you with this information.

For example, to display a list of commands for which HELP is available, type:

```
$ HELP RET
```

The system responds by displaying the list of HELP commands and prompting you for a choice of topic.

If you want information about a particular command, type that command after the topic prompt. For instance, if you want information about the PRINT command, type the following:

```
Topic? PRINT RET
```

The information displayed includes a synopsis of what the PRINT command does, the parameters it requires, and the qualifiers it can take.

If you want to know more about one of the PRINT command qualifiers, respond to the prompt "PRINT subtopic?" with that qualifier. (Remember to type the slash character (/) before the qualifier.) For example, to display information about the /COPIES qualifier of the PRINT command, type:

```
PRINT subtopic?/COPIES RET
```

If you know the subtopic on which you need help to begin with, you can simply type the following:

```
$ HELP PRINT/COPIES RET
```

When you have finished using HELP, press CTRL/Z. The dollar sign prompt is displayed in the left margin, indicating that VAX/VMS is ready to receive a command.

1.6.1 Exploring Several HELP Topics

The topics and commands you see when you enter the HELP command cover a wide range of subjects:

- To manipulate files, you can get information about the following DCL commands: COPY, DELETE, PURGE, RENAME, and TYPE. (Files are discussed in detail in Chapter 2.)
- For programming needs, you can get information about the LINK and RUN commands.
- For information about the available language compilers, enter the HELP command followed by the name of the compiler (for example, HELP BASIC, or HELP FORTRAN).
- For text processing needs, you can get information about available editors (HELP EDIT) and a text formatter (HELP RUNOFF).
- For operator needs, you can get information about the ALLOCATE, DEALLOCATE, MOUNT, and DISMOUNT commands.
- For information by category, enter the HELP HINTS command.

1.7 Using Utilities

VAX/VMS provides many different utilities. Each **utility** has the ability to perform tasks. You can think of a utility as an environment in which you can use a specific set of commands and/or qualifiers to perform the desired task.

To invoke a utility, you enter the corresponding DCL command. For example, to invoke the VAX/VMS Phone Utility (PHONE) you enter the DCL command PHONE. After you enter the DCL command and press RETURN, you will see the utility prompt. Some utilities have a prompt that matches their name. For example, the VAX/VMS Personal Mail Utility (MAIL) displays the following prompt:

```
MAIL>
```

Other utilities prompt you for a filename. For example, when you enter the DCL command SORT to invoke the VAX Sort Utility, you will see the following prompt:

```
_Input:
```

When you are using the type of utility that prompts you for a file name, you can add qualifiers to the DCL command string in order to tailor the utility to your own needs. The following example shows how to tailor the VAX/VMS Backup Utility (BACKUP) by specifying qualifiers on the command string when invoking the utility:

```
$ BACKUP/RECORD/IMAGE/LOG RET
_From:
```

When you are using the type of utility that displays a utility prompt (for example, MAIL>), you can enter utility commands at the utility prompt. Each utility responds to a different set of commands. For example, the VAX/VMS Phone Utility will recognize the following commands because they are PHONE commands:

- DIAL
- HOLD
- REJECT

If you invoke the VAX/VMS Personal Mail Utility (MAIL) and enter these same commands, you will see error messages displayed because these commands are not recognized by MAIL.

The following table lists three utilities, the DCL command you enter to invoke each utility, and the utility prompt:

Utility	DCL Command	Utility Prompt
MAIL	MAIL	MAIL>
PHONE	PHONE	%
MONITOR	MONITOR	MONITOR>

The following table lists four utilities, the DCL command you enter to invoke each utility, and the first DCL prompt displayed (which is a file name prompt):

Utility	DCL Command	DCL Prompt
BACKUP	BACKUP	_From:
MESSAGE	MESSAGE	_File:
PATCH	PATCH	_File:
SORT	SORT	_Input:

The following sections discuss two utilities in more detail: MAIL and PHONE. For more information about all the available utilities, see the *VAX/VMS Utilities Reference Volume*.

1.7.1 What is MAIL?

The VAX/VMS Personal Mail Utility (MAIL) allows you to send messages to other users on your system or on any other VAX computer that is connected to your system by means of DECnet–VAX. (For information about DECnet–VAX, see the *Guide to Networking on VAX/VMS*). To become familiar with MAIL, enter the MAIL command at the system prompt and use the twelve MAIL commands discussed in this section. These twelve commands will enable you to move around within MAIL. For more detailed information about MAIL, see the *VAX/VMS Utilities Reference Volume*.

These are the twelve MAIL commands discussed in this section:

SEND	DIRECTORY	EXTRACT
READ[/NEW]	DELETE	PRINT
FORWARD	MOVE	HELP
REPLY	SELECT	EXIT

1.7.1.1 Sending Mail

After you invoke MAIL, the first command to try is the SEND command. You can send a message to anyone on the system by entering their user name at the "To:" prompt. The following example shows how to send a message to a user named BACCI:

```
MAIL> SEND [RET]
To: BACCI [RET]
```

Try sending a message to yourself. Enter the SEND command at the MAIL> prompt and press RETURN. Enter your own user name at the prompt "To:" and press RETURN. Enter a subject when prompted and press RETURN again. The following example shows how to use the SEND command:

```
MAIL> SEND [RET]
To:     PIERCE [RET]
Subj:   Sailing [RET]
Enter your message below.  Press CTRL/Z when complete, or CTRL/C to quit:
```

When you finish entering the text of your message, press CTRL/Z. Because you are sending the message to yourself, MAIL will display a message on your screen announcing that you have new mail from yourself, like the following:

```
New mail on node FLAXEN from PIERCE
MAIL>
```

1.7.1.2 Reading Mail

Now, you are ready to use the READ command. To read the message you just sent to yourself, enter the READ command with the /NEW qualifier and press RETURN.

```
MAIL> READ/NEW
```

The only time you must specify the /NEW qualifier with the READ command is when you want to read your new mail that arrives while you are in the Mail Utility. If you are not in the Mail Utility and you receive new mail, invoke MAIL and enter the READ command (without the /NEW qualifier) to read the new message. Also, when you wish to reread old messages (messages that you have already read), enter the READ command. If you have just read a new message, and you want to reread an old message, enter the following command string:

```
MAIL> SELECT MAIL
```

(The MAIL command SELECT is discussed later in this section.) Now, you can use the READ command to reread the old message.

1.7.1.3 Forwarding Mail

You can forward a copy of a mail message to another user by entering the FORWARD command. MAIL will prompt you for the name of the user to receive the message. Try forwarding a copy of the message you just received back to yourself. Enter your own user name and press RETURN. Supply a subject when prompted and press RETURN. MAIL will signal that you have just received a new message. Enter the READ/NEW command to read the forwarded message.

1.7.1.4 Replying to Mail

When you receive a message and want to respond to it, enter the REPLY command and press RETURN. MAIL will display the header information as follows:

```
MAIL> REPLY
To:         FLAXEN::PIERCE
Subject:    Re:Sailing
Enter your message below.  Press CTRL/Z when complete, CTRL/C to quit:
```

When you finish typing your response, press CTRL/Z. Again, MAIL will signal that you have just received a new message. To read the message, enter the READ/NEW command.

1.7.1.5 Listing Mail Messages

When you want to see a list of all the new mail messages you have collected, enter the DIRECTORY command and press RETURN. MAIL will display a list like the following:

```
# From              Date                 Subject
1 FORBES            1-JUN-1985           How to Write a Memo
2 STELLA::BERT      2-JUN-1985           Using the Printer
3 FROST::BASTIEN    4-JUN-1985           Chicken Kiev
```

To see a list of all the new and old messages, enter the SELECT MAIL command string followed by the MAIL command DIRECTORY.

1.7.1.6 Organizing Mail into Folders

MAIL allows you to organize your messages by moving them into **folders**. By default, MAIL provides three folders: MAIL, NEWMAIL, and WASTEBASKET. The MAIL folder always exists. The NEWMAIL folder holds new mail messages before you read them. After you read a message it moves from the NEWMAIL folder to the MAIL folder. The WASTEBASKET folder is used to hold messages that you have deleted.

MAIL allows you to create your own folders as well. You can create as many folders as you want.

To move a message to a folder, enter the MOVE command (while you are reading the message) and press RETURN. MAIL will prompt you for a folder name. Type any name, for example, REVIEWS or JOKES or STATUS_REPORTS. When you enter the name of a new folder (one that you have not created before), MAIL asks whether or not you want to create it. Answer "y" . MAIL will also prompt you for a file name. You can specify the default mail file by pressing RETURN. A sample session demonstrating the MOVE command follows. (The name of the new folder is WINNERS and the default mail file is specified.)

```
MAIL> 2
MAIL> MOVE
_Folder: WINNERS
_FILE: RET
Folder WINNERS does not exist.
Do you want to create it (Y/N, default is N)?y
%MAIL-I-NEWFOLDER, folder WINNERS created
```

To move from one folder to another, use the SELECT command. If you want to move to the WINNERS folder, enter the following command string. (MAIL displays a message indicating the number of messages in the folder.)

```
MAIL> SELECT WINNERS
%MAIL-I-SELECTED, 1 message selected
```

To move to a folder named JOKES, enter the following command string:

```
MAIL> SELECT JOKES
%MAIL-I-SELECTED, 3 messages selected
```

To move to your default mail folder (MAIL), enter the command string: SELECT MAIL.

You can enter the DIRECTORY command to see a list of the messages in the folder you just selected. You can also enter the DIRECTORY/FOLDERS command to see a listing of your folders.

1.7.1.7 Deleting Mail

When you want to remove a message, use the DELETE command. You can either enter the DELETE command while you are reading the message or you can enter the DELETE command followed by the number of the message you want to remove. To remove the second message in the list, enter the following command string:

```
MAIL> DELETE 2
```

If you enter the DIRECTORY command after you have deleted a message (or messages), you will see the messages marked for deletion, as the following example shows:

```
# From              Date                Subject
1 FORBES            1-JUN-1985          How to Write a Memo
2 (Deleted)
3 FROST::BASTIEN    4-JUN-1985          Chicken Kiev
```

When you exit from MAIL, the messages marked for deletion will disappear.

1.7.1.8 Extracting Mail

When you want to move a mail message from your mail file to a sequential file that you can access from the DCL command level, use the EXTRACT command. Enter the EXTRACT command (while you are reading the message) and press RETURN. MAIL will prompt you for the name of a file. Then, when you exit from MAIL, the file will be listed in your main directory. The following example shows how to use the EXTRACT command to move a mail message to a file named GAMES.DAT.

```
MAIL> EXTRACT
_File: GAMES.DAT
%MAIL-I-CREATED, DISK:[BERGMAN]GAMES.DAT;1 created
MAIL>
```

1.7.1.9 Printing Mail

To make a hard copy of a mail message, enter the PRINT command while you are reading the message and press RETURN. (When you exit from MAIL, the system will respond by telling you that the message has entered the print queue.) The following example shows how to make a hard copy of message #4 by using the PRINT command:

```
MAIL> 4
    #4          4-AUG-1985 09:39:20           MAIL
From:  SPARTA::FELLINI
To:    MARSTON
Subj:  Rydell's Reasons
In reference to the meeting of July 26, I would like to explain
Rydell's opinion more fully...
MAIL> PRINT
MAIL> EXIT
     Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

1.7.1.10 Getting Help in Mail

To see detailed information about MAIL, enter the HELP command at the prompt MAIL>. Look at the HELP topic "Folders". "Folders" discusses the organization of the Mail Utility in detail. For more information about each MAIL command, you can keep using the HELP facility provided in MAIL or you can see the *VAX/VMS Utilities Reference Volume.*

1.7.1.11 Exiting from MAIL

When you are ready to leave MAIL, enter the EXIT command and press RETURN. Any messages marked for deletion will disappear. Any messages marked for printing will enter the print queue and a message similar to the following will be displayed:

```
MAIL> EXIT
  Job 790 entered on queue ATLAS_PRINT
```

1.7.2 What is PHONE?

The VAX/VMS Phone Utility (PHONE) allows you to "talk" with other users on your system or on any other VAX computer that is connected to your system by means of DECnet–VAX. It was designed to closely simulate the features of a real telephone.

Note: PHONE can be used only on video terminals with direct cursor positioning, such as the DIGITAL VT52, VT100, and VT200 Series terminals.

To become familiar with PHONE, enter the PHONE command at the system prompt and use the five commands discussed in this section. These commands will introduce you to PHONE. For more information about PHONE, see the *VAX/VMS Utilities Reference Volume*.

These are the five commands discussed in this section:

- ANSWER
- REJECT
- DIRECTORY
- HELP
- EXIT

1.7.2.1 How to Phone Another User

After you invoke PHONE, you can place a call to anyone who is currently logged on by entering their user name at the **switch hook character**, which by default is the percent sign (%).

Try phoning a friend on the system. Or, try phoning yourself. Enter your user name at the switch hook character (%) and press RETURN. Because you are calling yourself, your call will automatically be answered and PHONE will display the following message:

```
That person has answered your call.
```

Your screen display will divide showing the name of the person placing the call in the top section of the screen and the name of the person receiving the call in the bottom section of the screen. Figure 1–4 shows a PHONE screen display.

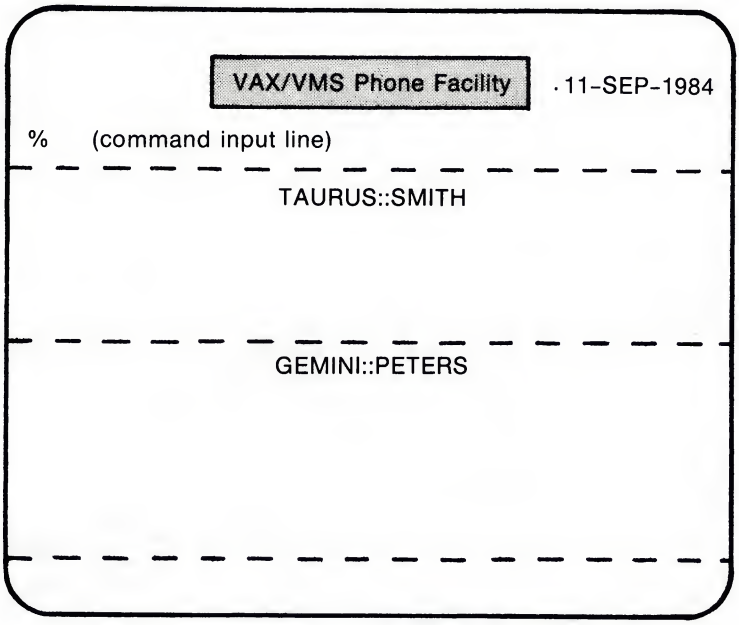
1.7.2.2 Answering a Phone Call

To answer a call from another user, invoke PHONE and enter the ANSWER command. Your screen display will divide and you can begin typing your conversation. Every character you type is displayed as part of the conversation (except the percent sign (%), which is the default switch hook character). While you are typing text, you can use the following key combinations to clear various parts of the screen:

Key combination	Function
CTRL/L	Clears all the text in your part of the screen.
CTRL/U	Clears the current line.
CTRL/W	Restores the entire screen.

To signal PHONE that you want to enter a PHONE command during a conversation, type the switch hook character (%) followed by the command.

Figure 1–4 Looking at a PHONE Screen Display



Note: The switch hook character is always displayed in column 1 of the command input line.

ZK-898-82

1.7.2.3 Rejecting a Phone Call

If you decide not to answer a call, PHONE keeps broadcasting the message on your screen until either the person calling cancels the call or you enter the REJECT command. When you enter the REJECT command, the user placing the call will receive a message at his terminal that the call has not been accepted.

1.7.2.4 Displaying a List of Users You May Call

To see a list of the users you may call, enter the DIRECTORY command. If you enter this command with a node name (as the following example shows), it lists the users on that system:

```
%DIRECTORY ATLAS
Press any key to cancel the directory listing and continue.
------------------------------------------------------------
Process Name     User Name        Terminal        Phone Status
LIPTON            LIPTON           TTE1:           available
Roy Rogers        ROGERS           VTA3:           available

2 persons listed.
```

1.7.2.5 Getting Help in PHONE

When you want general information about PHONE including descriptions for all the available PHONE commands, enter the HELP command and press RETURN.

1.7.2.6 Leaving PHONE

When you want to leave PHONE, enter the EXIT command at the switch hook character and press RETURN. Your entire screen will clear. You will see the DCL command prompt. For more detailed information about PHONE, see the *VAX/VMS Utilities Reference Volume*.

1.8 Logging Out

When you have finished your session at the terminal, use the LOGOUT command to end the terminal session:

```
$ LOGOUT RET
```

The system responds with the following:

```
MAYMON  logged out at 17-JUL-1982 12:43:10.38
```

Note that neither shutting off your terminal nor setting the REMOTE/LOCAL switch to LOCAL automatically causes you to **log out**. To ensure that you have logged out, you should use the LOGOUT command to end a terminal session. If you shut a terminal off without logging out properly, another user may be able to turn the terminal on later and use your account.

1.9 For More Information

The *VAX/VMS DCL Dictionary* is the primary reference manual for information about the DIGITAL Command Language. The manual contains complete descriptions of DCL commands, defines the grammar of the DCL command language, and illustrates command usage with many examples.

2 Working with Files

This chapter explains how to use DCL commands to manipulate files: how to identify, create, delete, and purge files; how to create and list directories; and how to print, copy, protect, and rename files.

2.1 What is a File?

A file contains information. One type of information a file can contain is text. For example, to organize a book, you can create a file for each chapter. To edit each chapter, you enter and manipulate text within each file. You might also create a file containing a memo. Then, you can use the Mail Utility to send the file containing the memo to other users. Or, you can edit the file to change the memo.

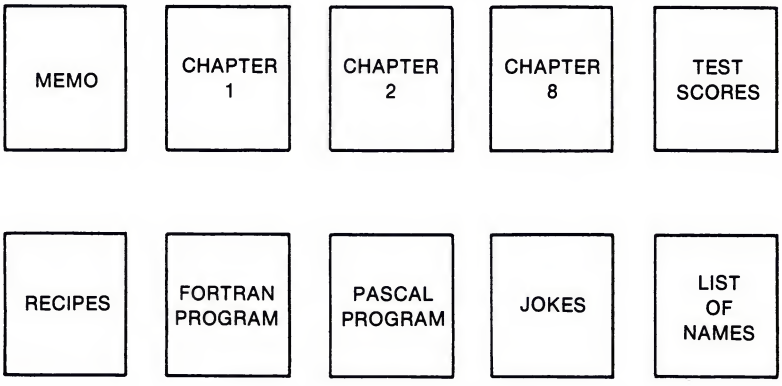
You can create a file and fill it with instructions using a language like FORTRAN or PASCAL. Then, you use a language compiler to generate a machine language that the computer will understand. This list of instructions is called a **program**. By entering the appropriate DCL commands and specifying the name of the file containing the program, you can make the computer perform the task.

If you find that you are often using the same series of DCL commands, you can create a file containing this series. (A series of DCL commands is called a **command procedure**. For information about command procedures, see Chapter 5.) Then, when you want to use this series of commands, you enter the appropriate DCL command and the name of the file containing the series.

You can create as many files as you want, filling each one with different types of information. The types of information can range from test scores to command procedures, from recipes to computer programs, from books to lists of names.

Figure 2–1 shows the types of information that ten different files can contain:

Figure 2–1 Types of Information that Files Can Contain



2.1.1 Looking at File Names, Types, and Versions

You can identify a file by specifying its **file name** and **file type** in the following format:

```
filename.type
```

The file name can have from one through 39 characters chosen from the letters A through Z, the numbers 0 through 9, and an underscore (_) or a dollar sign ($). When you create files, you can give them any names that are meaningful to you. A list of possible legal file names follows:

2409CHAP2
2409CHAP13
APPRAISE
BASIC_EXAMPLES
FORTRAN_EXAMPLES
GETTING_STARTED
MAIL
ORCHESTRA_MEMBERS
THINGS_TO_DO

The file type can be from 0 through 39 characters, and must be preceded by a period. Again, you can choose any of the letters A through Z or the numbers 0 through 9 for the file type. However, the file type usually describes more specifically the kind of data in the file. The system recognizes many default file types used for special purposes. Some of the more common default file types follow:

File Type	Use
COM	Command procedure
DAT	Data file
DIS	Distribution list for MAIL
EDT	Start-up command file for EDT editor
EXE	Executable program image file
JOU	Journal file used by the EDT editor
LIS	Output listing file
MAI	MAIL message file
MEM	Output file for DIGITAL Standard Runoff (DSR)
OBJ	Object module file output from a compiler or assembler
RNO	Input file for DIGITAL Standard Runoff (DSR)

Each high-level language has a default file type for source programs. Some of these file types are listed in the following table:

File Type	Contents
BAS	Input source file for the VAX BASIC compiler
B32	Input source file for the VAX BLISS-32 compiler
C	Input source file for the VAX C compiler
COB	Input source file for the VAX COBOL compiler
COR	Input source file for the VAX CORAL-66 compiler
FOR	Input source file for the VAX FORTRAN compiler
MAR	Input source file for the VAX MACRO assember
PAS	Input source file for the VAX PASCAL compiler
PLI	Input source file for the VAX PL/I compiler

The following list combines several previously mentioned file names with file types:

2409CHAP2.RNO
BASIC_EXAMPLES.BAS
MAIL.MAI
ORCHESTRA_MEMBERS.DIS
APPRAISE.REAL_ESTATE

In addition to a file name and type, every file has a version number that the system assigns to a file when the file is created or revised. When you initially create a file, the system assigns it a version number of 1. Subsequently, when you edit a file or create additional versions of it, the version number is automatically increased by one.

Version 12 of a file named APPRAISE.MEM follows:

```
APPRAISE.MEM;12
```

You rarely need to specify the version number with a file specification. The system assumes default values for version numbers. Version number defaults are determined as follows:

1. For an **input file**, the system uses the highest existing version number of the file. (Many system utilities take existing files, alter them, and produce new files. The existing file is called an input file and the newly produced file is called an output file.)

2. For an **output file**, the system adds 1 to the highest existing version number.

When you specify a version number in a file specification, precede the version number with a semicolon (;).

2.1.2 Using Wildcard Characters

A **wildcard character** is a symbol that you can use with many DCL commands to apply the command to several files at once, rather than specifying each file individually. Two wildcard characters, the asterisk (*) and the percent sign (%), can be used when you specify a file name and a file type. The asterisk can also be used to specify version numbers.

For example, you can specify all versions of a file by using an asterisk in place of the version number in the file specification. If, for example, you want to print all versions of the file TESTFILE.DAT without specifying each version number separately, enter the following command string:

```
$ PRINT testfile.dat;*
```

If there were no wildcard character in the above example, the PRINT command by default would apply only to the most recent version of the file TESTFILE.DAT.

The following command string prints all versions of all files with the file type of DAT:

```
$ PRINT *.dat;*
```

To print all versions of all files with the file name of TEST, enter the following command string:

```
$ PRINT test.*;*
```

The percent sign allows you to specify all files containing any single character in the position that the percent sign occupies in the file specification. For example, to print the latest version of several files with a file type of TXT and a file name that begins with CHAP but ends in a series of different numbers, as in CHAP1.TXT, CHAP2.TXT, and CHAP3.TXT, enter the following command string:

```
$ PRINT chap%.txt
```

Note that in this example the percent sign specifies only one character. Therefore, this PRINT command would not affect a file named CHAP.TXT or CHAPIX.TXT.

The following example shows how to print all the files beginning with the letters CHAP and having a file type of TXT:

```
$ PRINT CHAP*.TXT
```

2.2 How to Create a File

To create a file you can enter one of the following commands:

- EDIT
- CREATE

The EDIT command invokes an **editor** to create a file. The default VAX/VMS editor is EDT. To learn how to use the EDT editor, see the *Guide to Text Processing on VAX/VMS*.

You can also use the CREATE command to make a new file and specify the file name as a parameter. You can insert text immediately and terminate the insertion with <CTRL/Z>.

```
$ CREATE myfile.dat
This is the only line.
CTRL/Z
```

Unlike the EDIT command, the CREATE command does not modify an existing file.

2.3 How to Delete a File

Quite often, as you revise files you end up with many versions. Since these files take up space on your disk, you may want to delete versions of files that you no longer need.

The DELETE command deletes specific files. When you use the DELETE command, you must specify a file name, file type, and version number (having to specify a version number provides some protection against accidental deletion). However, any of these file components can be specified as a wildcard character. You can also enter more than one file specification in a command string by separating the file specifications with

commas. The following table shows some examples of the DELETE command:

Command	Result
$ DELETE average.obj;1	Deletes the file named AVERAGE.OBJ;1
$ DELETE *.lis;*	Deletes all files with file types of LIS (thus, this command deletes all versions of all program listings)
$ DELETE a.dat;1,a.dat;2	Deletes the first two versions of the same data file

2.4 How to Purge Your Files

When you find that you want to delete many versions of many files, you can use the PURGE command.

The PURGE command allows you to delete all but the most recent version of a file; therefore, no version number is permitted by the PURGE command. The following command string deletes all files named AVERAGE.FOR except the file with the highest version number:

```
$ PURGE average.for RET
```

Use the /KEEP qualifier with the PURGE command to specify that you want to keep more than one version of a file. The following command string deletes all but the two most recent versions of the file TEST.DAT:

```
$ PURGE/KEEP=2 test.dat RET
```

To clean up your entire directory, enter the PURGE command without any parameters or qualifiers.

2.5 How to Display a File at Your Terminal

The TYPE command displays a file at your terminal. To display a file named TEST.DAT, enter the following command string:

```
$ TYPE test.dat RET
This is the first line of a file created with the EDT editor.
```

While a file is being displayed, you can suspend and resume the upward movement, or **scrolling**, of the terminal display by using CTRL/S and CTRL/Q. To temporarily stop the display from scrolling, press CTRL/S; to continue the scrolling, press CTRL/Q. (The NO SCROLL key on VT100 terminals and the F1 key (Function Key 1) on VT200 Series terminals perform the same functions. Pressing NO SCROLL once suspends scrolling; pressing it again resumes scrolling.)

2.6 How to List Files in a Directory

A directory is a file that briefly catalogs a set of files. The directory includes the name, type, and version number of each file. (See Section 3.1.3 for more information about directories.)

To list the names of files in a particular directory, use the DIRECTORY command. When you enter the DIRECTORY command with no parameters or qualifiers, the command displays the files listed in your default directory on the terminal as the following example shows:

```
$ DIRECTORY RET
Directory BOOK2:[MALCOLM]    ❶

AVERAGE.EXE;2      AVERAGE.EXE;1      AVERAGE.FOR;2
AVERAGE.FOR;1      AVERAGE.OBJ;2      AVERAGE.OBJ;1    ❷

Total of  6 files.    ❸
```

❶ The disk and directory name (see Section 3.1.2 for information about disks)

❷ The file names, file types, and version numbers of each file in the directory

❸ The total number of files in the directory

When you enter the DIRECTORY command, you can provide one or more file specifications to obtain a listing of particular files. For example, to find out how many versions of the file AVERAGE.FOR currently exist, enter the DIRECTORY command as follows:

```
$ DIRECTORY average.for RET
Directory BOOK2:[MALCOLM]
AVERAGE.FOR;2    AVERAGE.FOR;1
Total of 2 files.
```

You can use the wildcard character (*) to display selective groups of files. If you want to see a list of all the files in your directory beginning with the letters "TR" , enter the following command string:

```
$ DIRECTORY TR* RET
Directory CIRCUS:[PERT]
TRAINS.DAT;16  TRAPEZE.BAR;2
TRAVEL.FUN;5
```

2.7 How to Print a File

To make a hard copy of a file, use the PRINT command. When you use the PRINT command to obtain a printed copy of a file, the system cannot always print the file immediately since there may be only one or two line printers for all users to share. The system enters the name of the file you want to print in a **queue**, and prints the file at the first opportunity.

A printed file is preceded by a **header page** describing the file so you can identify your own listing. For example, if you issue the following command string, the header page will show your user name and the file's name, type, and version number:

```
$ PRINT WRITERS:[jones]average.lis RET
  Job AVERAGE (queue GROUP_PRINT, entry 1995) started on BIG$LPA0
```

When you use the PRINT command, the system responds with a message indicating the job number it assigned to the print job.

The PRINT command also has qualifiers that allow you to control the number of copies of the file to print, the type of forms to print the file on, and so on. See the *VAX/VMS DCL Dictionary* for detailed information about these qualifiers.

2.8 How to Rename a File

To change the identification of one or more files, use the RENAME command. For example, the following command string changes the file name and type of the most recent version of the file PAYROLL.DAT to TEST.OLD:

```
$ RENAME payroll.dat test.old RET
```

After you enter the previous command string, the file name PAYROLL.DAT no longer exists. Its contents now reside in the file named TEST.OLD.

You can use the RENAME command to move a file from one directory to another. For example, the following command moves test.old from the directory [MALCOLM] to the subdirectory [MALCOLM.TESTFILES]:

```
$ RENAME [malcolm]test.old [malcolm.testfiles]
```

You can use wildcard characters if you want to change a number ofjfiles that have either a common file name or file type. The following command string changes the directory name for all versions of all files that have file names of PAYROLL:

```
$ RENAME payroll.*;* [malcolm.testfiles]*.*;* RET
```

The files are now cataloged in the subdirectory [MALCOLM.TESTFILES].

2.9 How to Protect a File

To prevent others from gaining unauthorized or undesired access to a file, enter the SET FILE/PROTECTION command specifying user categories with access types. The following tables list the four user categories and the four access types:

User Category	Type of User
OWNER	The user who created the file
GROUP	All users, including the owner, who have the same group number (or group identifier) in their user identification codes (UIC) as the owner of the file.
WORLD	All users
SYSTEM	All users who have system privilege (SYSPRV) or low group numbers (from 1 through 8)

Access Type	Type of Access
READ	The right to examine, print, or copy a file
WRITE	The right to modify or write a file
EXECUTE	The right to execute a file that contains executable program images
DELETE	The right to delete a file

A user identification code (UIC) can be either a pair of numbers or a name (or optionally, a pair of names). When a DCL command requires a UIC specification, you can use either format. However, the system translates all UICs to numbers when it determines a user's access.

A numeric UIC consists of a **group number** (g) and a **member number** (m) in the following format:

```
[g,m]
```

An example of two numeric UICs follow. (Notice that they both have the same group number):

```
[360,055]
[360,223]
```

A UIC can also be either one or two names. These names are called a **member identifier** and, optionally, a **group identifier**, as follows:

```
[member-identifier]
        or
[group-identifier,member-identifier]
```

An example of three UICs consisting of names follows. (Notice that the first two UICs have the same group identifier:

```
[JETS,BELLINI]
[JETS,FRANKLIN]
[RAMS,CHOO]
```

For detailed information about UICs, group numbers, member numbers, group identifiers, and member identifiers, see the *VAX/VMS DCL Dictionary* and the *Guide to Using DCL and Command Procedures on VAX/VMS.*

You can abbreviate user categories and access types to one letter, as the following table shows:

User Category	Access Type
O - OWNER	R - READ
G - GROUP	W - WRITE
W - WORLD	E - EXECUTE
S - SYSTEM	D - DELETE

When you use the SET FILE/PROTECTION command, separate the category from the type of access by a colon (:). Also, when you specify more than one category, use a comma (,) to separate one category from another and enclose the list of categories in parentheses ( ). For example, the following command string contains three categories, separated from each other by commas and enclosed in parentheses:

```
$ SET FILE/PROTECTION names.lis/protection=(O:RWE,G:RWE,W:R)
```

To summarize, there are four types of access: READ, WRITE, EXECUTE, and DELETE. And, there are four categories of users: SYSTEM, OWNER, GROUP, and WORLD. You can use the SET FILE/PROTECTION command, specifying a different type of access for each category of user. For example, if you want all users (WORLD category) to be able only to read a file named FRIENDS.DIS, enter the following command string:

```
$ SET FILE/PROTECTION friends.dis/protection=W:R
```

All other access is denied. Therefore, users will not be allowed to WRITE, EXECUTE, or DELETE the file named FRIENDS.DIS.

Or, if you want users in the GROUP category to be able to READ, but not DELETE a file named JOKES.COM, enter the following command string:

```
$ SET FILE/PROTECTION jokes.com/protection=G:R
```

If you want all users (WORLD category) to have total access to a file named GAMES.COM, enter the following command string:

```
$ SET FILE/PROTECTION games.com/protection=W:RWED
```

By default, when you create a file, it is protected in the following way:

- Both you (OWNER) and system users (SYSTEM) have total access, or the ability to READ, WRITE, EXECUTE, and DELETE the file. This protection information is coded in the following way:

  ```
  (SYSTEM:RWED,OWNER:RWED)
  ```

- Users in the GROUP category have READ and EXECUTE access, coded as follows:

  ```
  (GROUP:RE)
  ```

- Other users, who are not in your group, are considered part of the WORLD category and have no access, coded as follows:

  ```
  (WORLD: )
  ```

2.10 For More Information

For more detailed information about the commands presented here, see the *VAX/VMS DCL Dictionary*.

Remember, too, that while you are using the terminal, you can use the HELP command to receive assistance if you cannot remember a parameter or qualifier. Or, you can let the system prompt you for command parameters, if you cannot remember the order in which you have to enter them.

See the *Guide to Networking on VAX/VMS* for more information about networks and node specifications.

See the *Guide to VAX/VMS System Security* and the section on access control lists (ACL) in the *VAX/VMS Utilities Reference Volume* for detailed information on how to make your system secure.

3 Understanding Directory Structure

This chapter discusses the various parts of a full file specification.

3.1 Dissecting a Full File Specification

A complete file specification contains all the information the system needs to locate and identify a file. A complete file specification has the following format:

```
node::device:[directory]filename.type;version
```

You must supply the punctuation marks (colons, brackets, period, semicolon) to separate the components of the file specification.

Figure 3–1 shows the relationship between the parts of a full file specification. Notice how the file is listed in a directory, the directory resides on a device, and the device is located on a **node**.

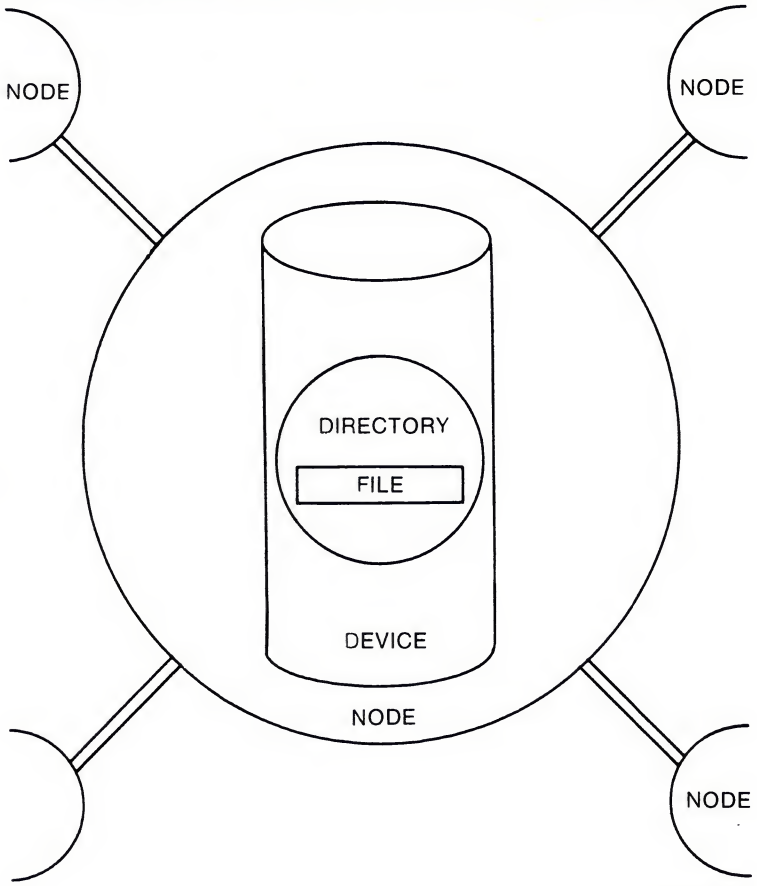
Figure 3-2, a full file specification, contains all the necessary information to enable the system to locate and identify the file STORIES.TXT in the [MCNALLY] directory, on device VAMP:, located on node DRACUL:.

Figure 3–3 shows the relationship between the various parts of the previous file specification.

3.1.1 Looking at Nodes

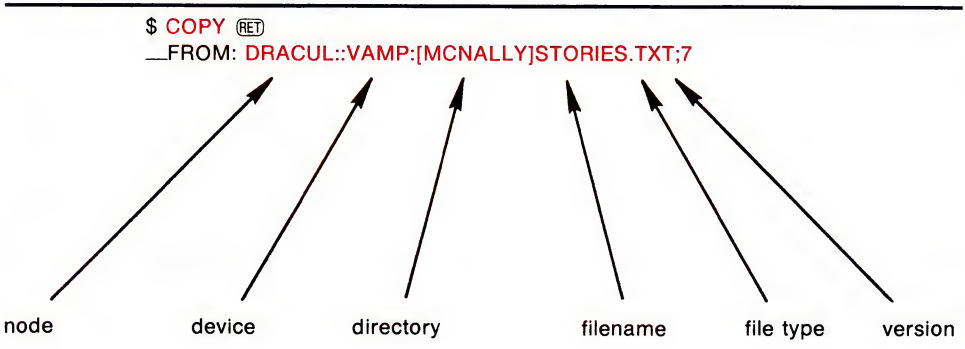
When computer systems are linked together, they form a **network**. Each system in the network is called a node, and is identified within the network by a unique node name. Your system may or may not be part of a larger network. For detailed information about networks and nodes, see the *Guide to Networking on VAX/VMS*.

Figure 3–1 Relationship between Parts of Full File Specification



If your system is a network node, you may be able to gain access to a file located at another node on the network by adding a **node specification** to the first part of the file specification. (This specification will allow you access to the file only if the user owning the file has permitted other users access to it.)

Figure 3–2 Full File Specification



For example, to access a file named ZAP.LIS, which is listed in the [LAWRENCE] directory, stored on device DEVO: on node LOTUS, enter the following command string:

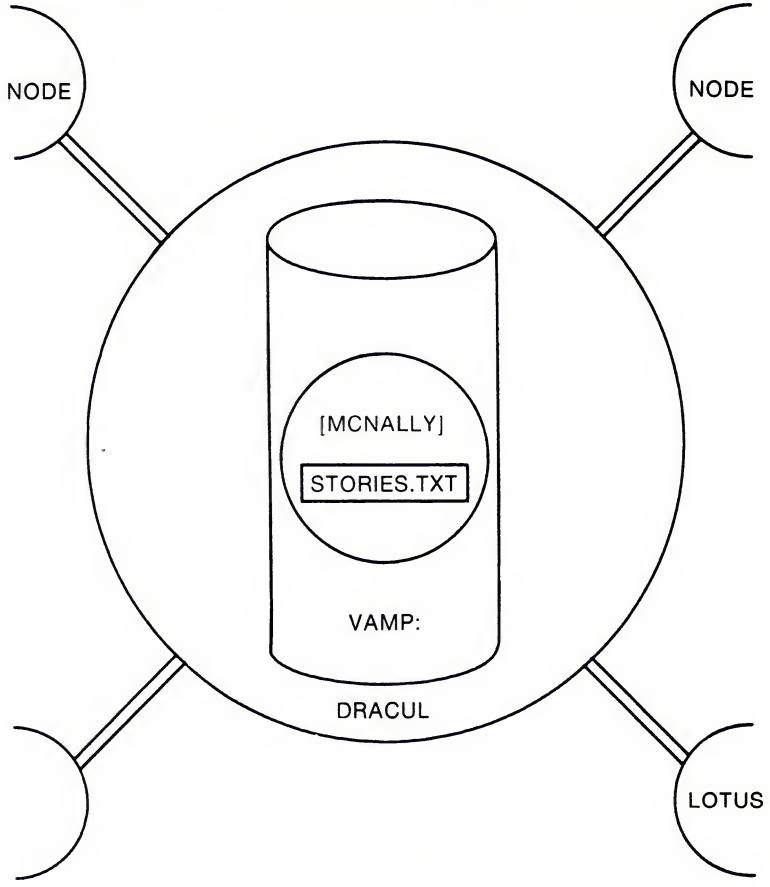
```
$ TYPE LOTUS::DEVO:[LAWRENCE]ZAP.LIS
```

Figure 3–4 shows the relationship between the various parts of the previous file specification.

If you do not specify a node, the system assumes by default that the file belongs to your own, or local, node.

You should use network defaults when possible. For example, when you access a file on you own local node, you do not need to specify the node name. By not specifying the node name, you avoid the unnecessary overhead of invoked network routines.

Figure 3–3 Relationship between Parts of DRACUL::VAMP:[MCNALLY]STORIES.TXT

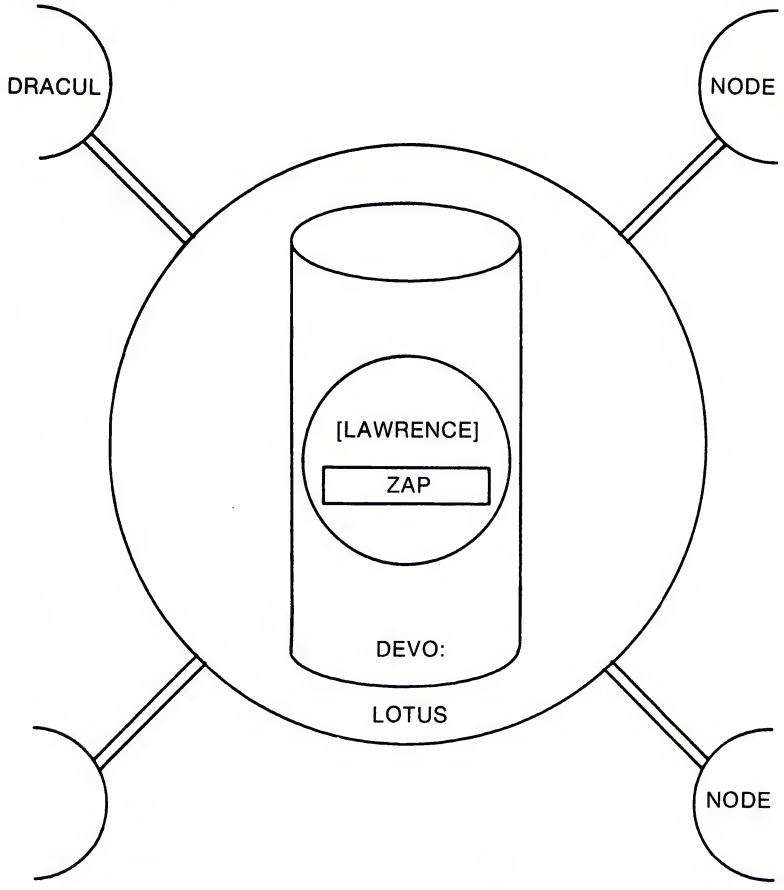


3.1.2 Looking at Devices

The second part of a file specification, the device name, identifies the physical device on which a file is stored. **Tapes** and disks are examples of devices. A device name has the following three parts:

- The device type, which identifies the hardware device. (For example, an RP06 disk is DB and a TE16 magnetic tape is MT.)

Figure 3–4 Relationship between Parts of LOTUS::DEVO:[LAWRENCE]ZAP.LIS



- A controller designator, which identifies the hardware controller to which the device is attached.
- The unit number, which uniquely identifies a device on a particular controller.

Several examples of device names follow:

Name	Device
DBA2	RP06 disk on controller A, unit 2
MTA0	TE16 magnetic tape on controller A, unit 0
TTB3	Terminal on controller B, unit 3

You may notice that your device names are words instead of four-digit codes. These words are logical names. (Section 3.2 explains logical names.)

For example, if you enter the DCL command SHOW DEFAULT and see your default device displayed like the following, you will know that your system manager has set up logical names to indicate the devices available to you:

```
$ SHOW DEFAULT
 BOOK1:[HARVEY]
```

In this example, the logical device name is BOOK1:.

You can use these logical names when referring to files to achieve file and device independence. If you specify a file using a logical device name, you can access the file regardless of which physical device holds the disk or tape containing your file. Your system manager will ensure that the logical device names are always equated to the correct physical devices.

If you want to access a file that is located on the same node as your own, but is stored on a different device, you must specify the device name as part of the file specification. (When you use a logical name in a file specification, you must terminate the name with a colon.) The following example shows how to access a file named TREES.DAT, which is stored on a magnetic tape labeled TAPE1:

```
$ TYPE TAPE1:TREES.DAT
```

You do not need to specify a node name in the previous command string because the file TREES.DAT is located on your own node.

If you omit a device name from a file specification, the system supplies the default value; that is, it assumes the file is on the disk assigned to you when the system manager set up your account. This disk is your **default disk**.

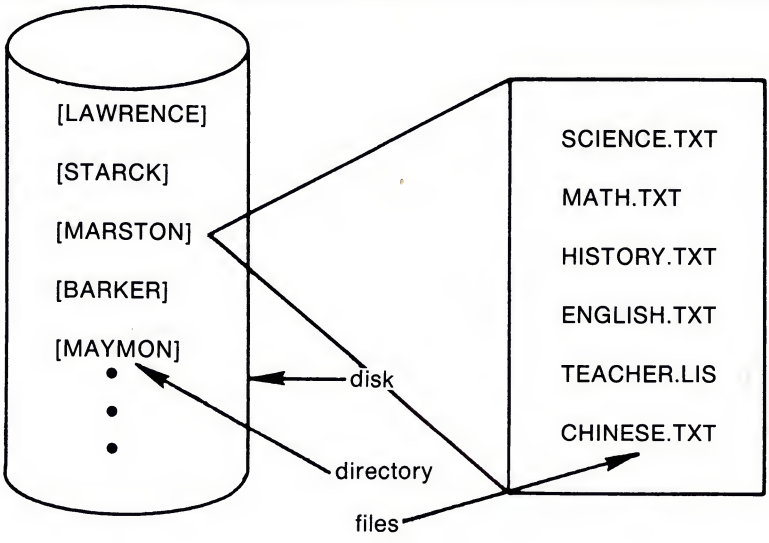
To list all the devices on the system, enter the SHOW DEVICES command.

3.1.3 Looking at Directories

Since a disk can contain files belonging to many different users, each user of a given disk has a directory that catalogs all the files belonging to him or her on that device. A directory is a file that catalogs other files. A directory file contains the names and locations of files in a format that only the system understands.

Figure 3–5 shows a disk (TEACHERS1:) containing a list of five directories (for users LAWRENCE, STARCK, MARSTON, BARKER, and MAYMON) and the files listed in the [MARSTON] directory:

Figure 3–5 Files in [MARSTON] Directory



To access the file SCIENCE.TXT, you would enter the following command string:

```
$ TYPE TEACHERS1:[MARSTON]SCIENCE.DAT
```

As with the default disk, if you do not specify another directory, or if you do not specify any directory, the system applies the default; it assumes that the files to which you refer are cataloged in your default directory. You can find out what your current default disk and directory are by issuing a SHOW DEFAULT command:

```
$ SHOW DEFAULT
TEACHERS1:[MALCOLM]
```

The system's response to the SHOW DEFAULT command indicates that the user's default device is TEACHERS1: and the default directory is [MALCOLM].

To gain access to files in other directories (including directories that catalog files belonging to other users), specify the directory name in a file specification. For example, to display the contents of a file named CONTENTS.DAT belonging to a user whose directory is [JONES], issue the TYPE command as shown below:

```
$ TYPE [JONES]CONTENTS.DAT RET
.
.
.
```

Note that the file specification does not include a device name. For the TYPE command to execute successfully, the directory [JONES] must be on your default disk device. This is because the system always applies a default when you omit a device name. If user JONES's directory is on the disk RESEARCH3:, you would issue the TYPE command as follows:

```
$ TYPE RESEARCH3:[JONES]CONTENTS.DAT RET
.
.
.
```

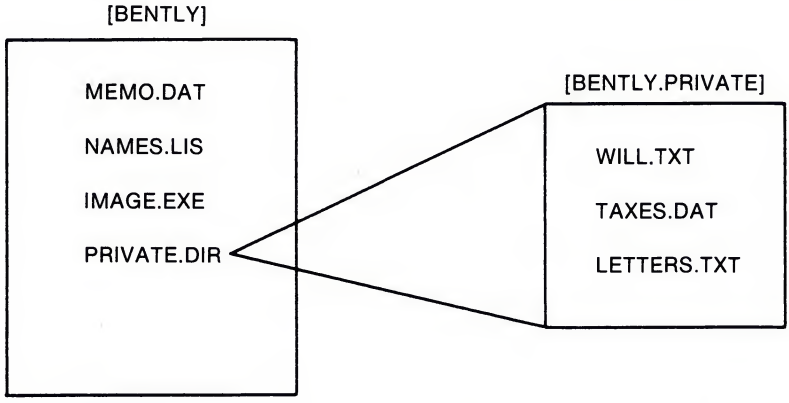
In both of these examples, it is assumed that the user JONES has given other users access to files in the directory. You can explicitly allow or restrict access to your own files, either generally or on a file-by-file basis, with the SET PROTECTION command. See the *VAX/VMS DCL Dictionary* for more information about directory and file protection and for a detailed description of the SET PROTECTION command.

3.1.4 Looking at Subdirectories

Files can also be cataloged in **subdirectories**. A subdirectory is a file (cataloged in a higher directory) that contains additional files.

Figure 3–6 shows a directory [BENTLY] containing one subdirectory [BENTLY.PRIVATE]:

Figure 3–6 Files in [BENTLY.PRIVATE] Subdirectory



ZK-1625-84

A subdirectory name is formed by **concatenating** (or joining) its name to the name of the directory that lists it and separating the names with a period. The following TYPE command requests a display of the file MEMO.SUM, which is cataloged in the subdirectory [JONES.DATAFILES]. The subdirectory file name is DATAFILES.DIR, and is cataloged in the directory [JONES]:

```
$ TYPE [jones.datafiles]memo.sum
```

You can use subdirectories to organize and separate your files.

3.1.4.1 How to Create a Subdirectory

Normally, the system manager provides each system user with one directory in which to maintain files. If you are a frequent user of the system and work with several applications, you may find it convenient to create several subdirectories, cataloging them in your main directory. You can create subdirectories in any directory in which you can create files.

Use the CREATE/DIRECTORY command to create a subdirectory. The following command creates the subdirectory file TESTFILES.DIR in the directory [MALCOLM], resulting in a subdirectory with the name [MALCOLM.TESTFILES]:

```
$ CREATE/DIRECTORY [malcolm.testfiles]
```

You can specify the subdirectory name [MALCOLM.TESTFILES] in commands or programs.

3.1.4.2 Changing Your Default Directory

To establish another directory or subdirectory as your default directory, use the SET DEFAULT command. The following example shows how to create a new file in the subdirectory [MALCOLM.TESTFILES] by changing your default directory and then creating the file with the EDIT command:

```
$ SET DEFAULT [malcolm.testfiles]
$ EDIT newfile.txt
Input file does not exist
[EOB]
*
```

The new file will be cataloged in the subdirectory [MALCOLM.TESTFILES]. (You could also do this by specifying the subdirectory as part of the file specification when you use the EDIT command.)

You can also use the SET DEFAULT command to change your default disk. The following example shows how to specify PILOT: as your default disk:

```
$ SET DEFAULT PILOT:
```

After you issue this command, the system uses the disk PILOT: as the default disk for all files that you access or create.

You can change your default disk and directory as often as is convenient. The changes you make with the SET DEFAULT command remain in effect until you either issue another SET DEFAULT command or log out.

3.2 Using Logical Names to Save Time

A **logical name** is a name that is equated to an equivalence string, or to a list of equivalence strings. An equivalence string can be any group of characters. Most often an equivalence string is a file specification, a device name, or another logical name. You use logical names for the following purposes:

- To reduce typing by using logical names as a short way of specifying files or directories that you refer to frequently.
- To avoid confusion about the location of volumes.
- To keep your programs and command procedures independent of physical file specifications.

You can use either the DEFINE command or the ASSIGN command to create a logical name by associating it with another name, the equivalence name. (The DEFINE and ASSIGN commands require different syntax. This chapter will demonstrate logical names with the DEFINE command. For information about the ASSIGN command, see the *VAX/VMS DCL Dictionary*.)

The following command assigns the logical name ZAP to the file specification DRACUL::DOC1:[MALVOLIO]ZAPISTS.DAT;21:

```
$ DEFINE ZAP DRACUL::DOC1:[MALVOLIO]ZAPISTS.DAT;21
```

After you have entered this command string, you can use the logical name ZAP in place of the longer file specification. The following command displays the contents of the file:

```
$ TYPE ZAP
```

This section discusses how to use logical names.

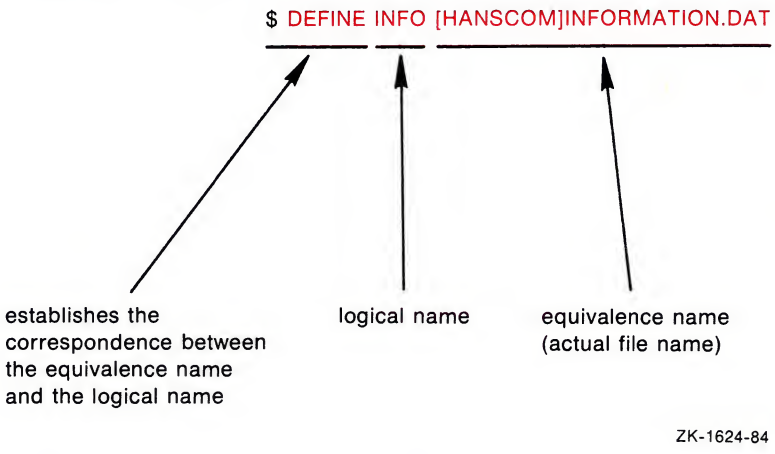
3.2.1 How to Use Logical Names

You can use the following formula to assign a logical name:

```
$ DEFINE logical-name equivalence-name
```

For example, Figure 3-7 assigns the logical name INFO to a file named INFORMATION.DAT, which is located in a directory named [HANSCOM]:

Figure 3–7 Assigning a Logical Name



When you want to access the file INFORMATION.DAT (in the [HANSCOM] directory), you can use the logical name INFO. To display the file, enter the following command string:

```
$ TYPE INFO
    .
    .
    .
```

When you create a logical name, it is maintained in a **logical name table**. A logical name table contains a set of logical names and their equivalence names. For detailed information about logical name tables, see the *VAX/VMS DCL Dictionary*.

You can use the SHOW LOGICAL command to display a logical name and its equivalence name. For example, to display the logical name HOME, you would enter the following command:

```
$ SHOW LOGICAL HOME
    .
    .
    .
```

The system searches the logical name tables for the logical name HOME. If it finds an entry, it displays the logical name and its equivalence name, and identifies the logical name table in which it found the logical name. In this example, the logical name HOME occurs in the process logical name table with the equivalence name of BOOK2:[JACK]. The system displays the following information:

```
0 "HOME" [super] = "BOOK2:[JACK]" (LNM$PROCESS_TABLE)
```

The following example shows how the DEFINE command equates the logical name MYFILE to the file PERSONNEL.REC listed in the directory CHUCK. To display this file on the terminal, enter the TYPE command as follows:

```
$ DEFINE myfile [chuck]personnel.rec
$ TYPE myfile
```

A logical name can also define only the first portion of a file specification. The following example shows how the DEFINE command equates the logical name TEST to the disk, device, and directory SCIENCE4:[MALCOLM.TESTFILES]. Subsequently, the RUN command executes the program image MEMO.EXE cataloged in this subdirectory and the PRINT command prints another file:

```
$ DEFINE test SCIENCE4:[malcolm.testfiles]
$ RUN test:memo
$ PRINT test:memo.lis
```

The system always examines file specifications to see if the portion of the file specification that precedes the colon (:) is a logical name; if it is (as in this example), the system substitutes the equivalence name.

3.2.2 What are System Default Logical Names?

When you log in to the system or submit a batch job, the system provides several default logical names. These names are used by the **command interpreter** to read your commands and to print responses or error messages.

Most system default logical names have the following format:

```
xxx$name
```

The three-character prefix, xxx, indentifies the system component that uses the logical name. The use of a dollar sign ($) within logical names is reserved for DIGITAL.

Several system default logical names follow:

Name	Use
SYS$COMMAND	The default device name of your terminal.
SYS$INPUT	The default input stream from which the system reads commands and your programs read data. The default interactive assignment for SYS$INPUT is your terminal. The default batch assignment for SYS$INPUT is the command procedure or batch stream.
SYS$OUTPUT	The default output stream to which the system writes responses to commands and your programs write data. The default interactive assignment for SYS$OUTPUT is your terminal. The default batch assignment for SYS$OUTPUT is the batch job log file.
SYS$ERROR	The default device to which the system writes all error and informational messages. The default interactive assignment for SYS$ERROR is your terminal. The default batch assignment for SYS$ERROR is the batch job log file.
SYS$DISK	Your default disk device. The default assignment is initially set in your User Authorization File (UAF) and can be changed with the DCL command SET DEFAULT.

Issue the following command to find out the equivalence names for these and other logical names created for your process:

```
$ SHOW LOGICAL
```

You may want to redefine SYS$OUTPUT to redirect output from your default device to a file. For example, if you want a hard copy of an online HELP file, you can assign an equivalence name (for example, HELP_LOGICAL_EXAMPLES.DAT) to the logical name SYS$OUTPUT, rechanneling output from your terminal to a file. Then, as you enter DCL commands interactively, the output will go to the specified file (equivalence name). To revert output back to your terminal (away from the file), enter the DEASSIGN command. The following example demonstrates how to make a hard copy of the online HELP examples for the DCL command SHOW LOGICAL:

```
$ DEFINE SYS$OUTPUT HELP_LOGICAL_EXAMPLES.DAT
$ HELP SHOW LOGICAL EXAMPLES
Topic? RET
$ DEASSIGN SYS$OUTPUT
$ PRINT HELP_LOGICAL_EXAMPLES.DAT
```

If you want to capture the output from only one DCL command, use the /USER_MODE qualifier as follows:

```
$ DEFINE/USER_MODE SYS$OUTPUT HELP_LOGICAL_EXAMPLES.DAT
$ HELP SHOW LOGICAL EXAMPLES
Topic? RET
$ PRINT HELP_LOGICAL_EXAMPLES.DAT
```

When you use the /USER_MODE qualifier, you do not need to enter the DEASSIGN command to revert the output back to your terminal. (For more detailed information, see the description of the DEFINE command in the *VAX/VMS DCL Dictionary*.)

You can also use these logical names in programs. For example, if you code a program to write a file to a device named SYS$OUTPUT, the output file goes to your terminal if you execute the program interactively, or to the batch job log file if you execute the program in a batch job.

4 Program Development

Four steps are required to develop a program:

- Creating the **source program** file
- Compiling or assembling the source program file to produce an **object module** file
- Linking the object module file to produce an **image**
- Executing and debugging the program

You should be familiar with a text editor to create and debug a source program file. The default editor for VAX/VMS is EDT. To invoke the EDT editor, you enter the DCL command EDIT. The *Guide to Text Processing on VAX/VMS* describes how to use EDT. For complete descriptions of all the available EDT commands, see the *VAX EDT Reference Manual.*

4.1 Creating the Program

In order to run your program, you must first create a file of the program source statements. The default file type corresponds to the language in which the program is written. For instance, if your program is written in VAX BASIC, its file type default is BAS. Table 4–1 lists the default file types for source program files written in several VAX languages.

Table 4–1 Default File Types for Source Program Files

File Type	Input Source File for:
BAS	VAX BASIC compiler
B32	VAX BLISS-32 compiler
C	VAX C compiler
COB	VAX COBOL compiler
COR	VAX CORAL-66 compiler

Table 4–1 (Cont.) Default File Types for Source Program Files

File Type	Input Source File for:
FOR	VAX FORTRAN compiler
MAR	VAX MACRO assembler
PAS	VAX PASCAL compiler
PLI	VAX PL/I compiler

4.2 Compiling or Assembling the Program

To prepare your source program for execution by the computer, a language processor must translate it into a format that the computer can read. That is, your program must be either assembled or compiled, depending upon whether it is written in assembly language or in one of the high-level languages supported by VAX/VMS.

Both **compilers** and **assemblers** are programs that translate source programs into binary **machine code** that can be interpreted by the computer. An **assembly language** is usually designed for a specific computer, and it generally assembles line for line into machine code. Most high-level languages, on the other hand, are designed to be universal, and usually compile one line of source code into several lines of machine code. If your source program is written in assembly language (in this case, VAXhMACRO), you invoke the VAX MACRO assembler to translate it. If it is written in a high-level language (such as BASIC, C, COBOL, FORTRAN, PASCAL, or PL/I), you invoke the appropriate VAX language compilers to compile the program.

Table 4–2 lists the DCL commands you use to invoke various language processors.

Table 4–2 DCL Commands to Invoke Language Processors

Command	Language Processor Invoked
BASIC	VAX BASIC compiler
BLISS	VAX BLISS-32 compiler
CC	VAX C compiler
COBOL	VAX COBOL compiler
CORAL	VAX CORAL-66 compiler
FORTRAN	VAX FORTRAN compiler
MACRO	VAX MACRO assembler
PASCAL	VAX PASCAL compiler
PLI	VAX PL/I compiler

Each of these commands invokes a compiler (or assembler) to translate the source program named in the file that follows the command. Although each command differs slightly in its parameters and qualifiers, the command format is essentially the same:

```
$ FORTRAN myfile
```

This command invokes the FORTRAN compiler to translate the file MYFILE into machine code, writing it to an output file called an object module. Since no file type is specified, the compiler assumes the default file type of FOR.

4.3 Linking the Object Module

An object module is not, in itself, executable; generally, it contains references to other programs or routines that must be combined with the object module before it can be executed. It is the function of the **linker** to do the combining.

The LINK command invokes the VAX Linker. (For detailed information about the VAX Linker, see the *VAX/VMS Utilities Reference Volume*.) The linker searches system libraries to resolve references to routines or symbols that are not defined within the object modules it is linking. You can request the linker to include more than one object module as input, or specify your own libraries of object modules for it to search. The format of the LINK command is:

```
$ LINK myfile
```

Since no file type is specified, the linker supplies a default file type of OBJ for object modules.

The linker creates an image, which is a file containing your program in an executable format. An image file has a default file type of EXE.

4.4 Executing the Program

The RUN command executes an image, that is, it places the image created by the linker into memory so that it can run. The format of the RUN command is:

```
$ RUN myfile
```

Since no file type is specified, the RUN command uses the default file type of EXE for executable images.

The first time you run a program, it may not execute properly; if it has a bug or programming error, you may be able to determine the cause of the error by examining the output from the program. When you have determined the cause of the error, you can correct your source program and repeat the compile, link, and run steps to test the result. Figure 4–1 illustrates these steps in program development.

Figure 4–2 lists the four steps you follow to develop a program using the BASIC language processor. (The name of the file containing the program is PROG.BAS.)

4.5 Looking at Sample Programs

These sections illustrate the steps of program development with three sample programs: a BASIC example for novice users, a MACRO example for assembly language users, and a FORTRAN example for high-level language users. These sections describe the input and output files used in each step and the naming conventions for the files. They also present optional command qualifiers you can use to create additional output files, including program listings. If you have access to a terminal, you can create the programs and issue the commands that are described.

Figure 4–1 Program Development

Use the *editor* to create a disk file containing your source program statements. Specify the name of this file when you invoke the compiler or assembler.

Commands invoke language processors that check syntax, create object modules, and if requested, generate program listings.

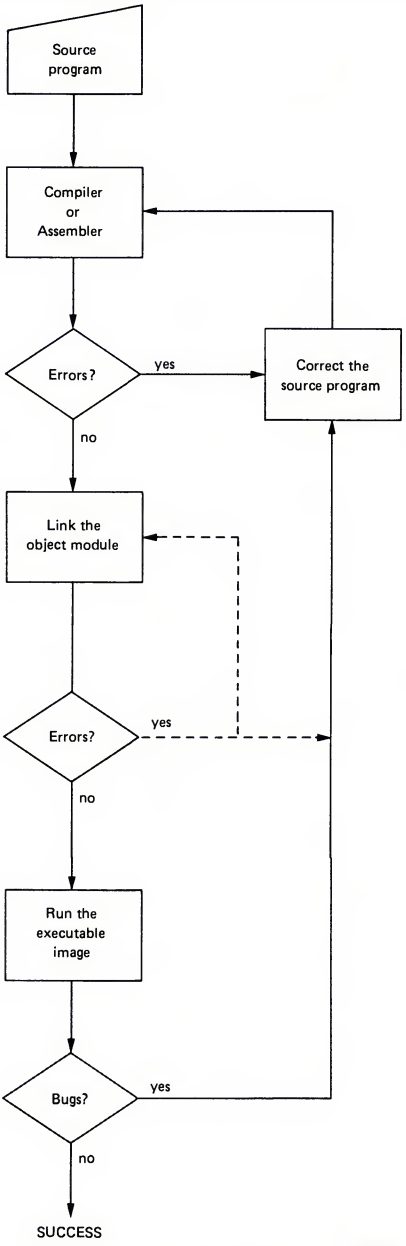
If a processor signals any errors, use the editor to correct the source program.

The *linker* searches the system libraries to resolve references in the object module and create an executable image. Optionally, you can specify private libraries to search, and request the linker to create a storage map of your program.

The linker issues diagnostic messages if an object module refers to subroutines or symbols that are not available or undefined. If the linker cannot locate a subroutine, you must reissue the *LINK* command specifying the modules or libraries to include. If a symbol is undefined, you may need to correct the source program.

The *RUN* command executes a program image. While your program is running, the system may detect errors and issue messages. To determine if your program is error-free, check its output.

If there is a bug in your program, determine the cause of the error and correct the source program.



ZK-763-82

Figure 4–2 Four Steps in Program Development

What You Do	Command Line You Enter	Input File You Supply	Resulting Output File
Create a source program file	$ EDIT prog.bas	prog.bas	prog.bas
Compile the source program to produce an object module file	$ BASIC prog.bas	prog.bas	prog.obj
Link the object module file to produce an image	$ LINK prog.obj	prog.obj	prog.exe
Run the executable image	$ RUN prog.exe	prog.exe	-

4.5.1 An Introductory BASIC Program

This section describes four steps you can follow to develop a BASIC program that will add three integers:

1 Use an editor to create a file named ADD.BAS containing the following six lines:

```
10      INPUT "What is the first integer" ;B
20      INPUT "What is the second integer" ;C
30      INPUT "What is the third integer" ;D
40      A = B + C + D
50      PRINT "Their sum is" ;A
60      END
```

2 Compile the source program file (ADD.BAS) to produce an object module file (ADD.OBJ):

```
$ BASIC ADD.BAS
```

3 Link the object module file (ADD.OBJ) to produce an image file (ADD.EXE):

```
$ LINK ADD.OBJ
```

4 Run the executable image:

```
$ RUN ADD.EXE
```

When you enter the RUN command, the ADD program will prompt you for three integers and give you their sum.

4.5.2 A FORTRAN Program

The steps required to prepare a VAX FORTRAN program to run on VAX/VMS are illustrated in Figure 4–3. Figure 4–3 also notes the default file types used by the FORTRAN, LINK, and RUN commands. For any of these commands, you can specify an explicit file type to override the defaults when you name an input or output file.

Note: The VAX FORTRAN compiler is referred to simply as FORTRAN throughout this guide.

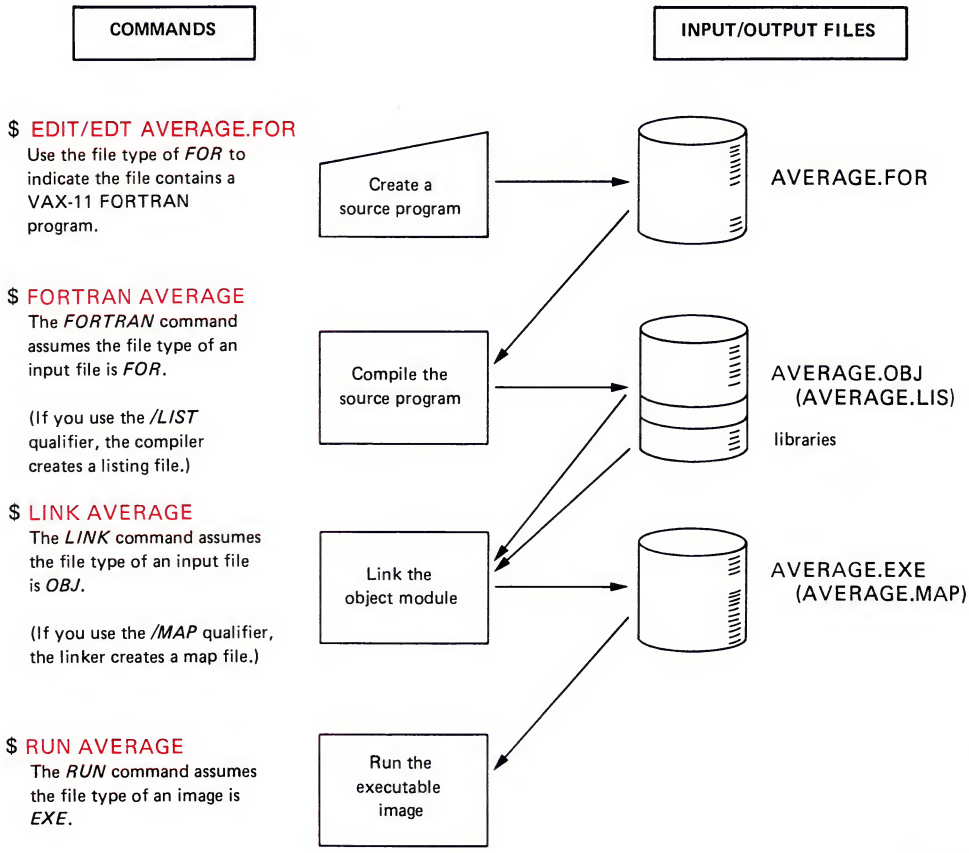
4.5.2.1 Creating the Source Program

Use an editor to create a source program interactively. For example, to create the FORTRAN program called AVERAGE, you can issue the DCL command EDIT. The EDIT command invokes EDT, which is the default editor for VAX/VMS:

```
$ EDIT average.for
```

The program AVERAGE follows. When you type the input statements, you can use the TAB key to align the statement and comments columns.

Figure 4–3 Commands for FORTRAN Program Development



```
      PROGRAM AVERAGE
C     COMPUTES THE AVERAGE OF NUMBERS ENTERED AT TERMINAL
C     TO TERMINATE THE PROGRAM, ENTER 9999
      TOTAL = 0                    ! INITIALIZE ACCUMULATOR
      N = 0                        ! INITIALIZE COUNTER
5     N = N + 1
      WRITE (6,10)                 ! PROMPT TO ENTER NUMBER
10    FORMAT (' ENTER NUMBER, END WITH 9999')
      READ (5,20) K                ! READ NUMBER FROM TERMINAL
20    FORMAT I10
      IF (K .EQ. 9999) GOTO 40     ! 9999 MEANS NO MORE INPUT
      TOTAL = TOTAL + K            ! COMPUTE TOTAL WITH NUMBER
      GOTO 5
C     NOW, COMPUTE AVERAGE BY DIVIDING TOTAL BY THE NUMBER OF
C     TIMES THROUGH THE LOOP
40    AVERAG = TOTAL/N
      WRITE (6,50) AVERAG          ! DISPLAY THE RESULT
50    FORMAT ('       AVERAGE IS  ',F10.2)
      STOP
      END
```

The program AVERAGE reads and writes lines to the current input and output devices; it prompts you to enter numbers and then computes the average of the numbers entered. The program AVERAGE purposely has a syntax error and a bug, so you can get an idea of how to use VAX/VMS to debug a program.

4.5.2.2 The FORTRAN Command

When you enter the FORTRAN command from the terminal, the FORTRAN compiler does the following by default:

- Produces an object module that has the same file name as the source file and a file type of OBJ
- Uses FORTRAN compiler defaults when it creates the output files (qualifiers in the FORTRAN command string can override these defaults)

To compile the source program AVERAGE, issue the following command:

```
$ FORTRAN average
```

Since the FORTRAN command assumes a file type of FOR, you need not specify the file type when you name the file to be compiled.

If the compilation is successful—that is, if the compiler did not detect any errors—the system displays a prompt for the next command as follows:

```
$
```

If there are any errors, the FORTRAN compiler displays information on the terminal. If you entered the source program AVERAGE exactly as it appeared above, then you received the following messages:

```
%FORT-F-ERROR 33, Missing operator or delimiter symbol
    [FORMAT I] in module AVERAGE at line 15
%FORT-F-ENDNOOBJ, TEST2:[MALCOLM]AVERAGE.FOR;1,
      completed with 1 diagnostic-object deleted
```

These fatal error messages indicate that the FORMAT statement was incorrectly coded; you must put parentheses around the format specification.

To correct the error, edit the following line in the source file:

```
20   FORMAT I10
```

The corrected line, which contains parentheses, follows:

```
20   FORMAT (I10)
```

Now you can recompile the program:

```
$ FORTRAN average
```

The FORTRAN command always uses, by default, the version of the file with the highest version number. If the program compiles successfully this time, you can go on to the next step. Otherwise, repeat the procedure of correcting the source file and compiling it.

When you compile a source program, use the /LIST qualifier with the FORTRAN command to request the compiler to create a program listing. For example:

```
$ FORTRAN/LIST average
```

The FORTRAN compiler creates, in addition to an object module, a file named AVERAGE.LIS. To obtain a printed copy of the program, use the PRINT command as shown below:

```
$ PRINT average
```

The PRINT command uses the default file type of LIS.

4.5.2.3 Linking the Object Module

To link the program AVERAGE, issue the LINK command as follows:

```
$ LINK average
```

This LINK command creates a file named AVERAGE.EXE, which is an executable program image. The linker automatically includes in the executable image any library routines that the compiler requested for input/output handling, error routines, and so on.

4.5.2.4 Running the Program

To execute the program AVERAGE, use the RUN command. When you issue the RUN command, you provide the name of an executable image. By default, the RUN command assumes a file type of EXE. Thus, to run the program AVERAGE, type the RUN command as follows:

```
$ RUN average
```

AVERAGE is interactive: it prompts you to continue entering numbers and it keeps a cumulative sum of the numbers you enter. When you enter 9999, it computes the average of all the numbers you entered. A typical run of this program might appear as follows:

```
ENTER NUMBER, END WITH 9999
33<RET>
ENTER NUMBER, END WITH 9999
66<RET>
ENTER NUMBER; END WITH 9999
99<RET>
ENTER NUMBER, END WITH 9999
9999<RET>
AVERAGE IS   49.50
FORTRAN STOP
$
```

As you can see, the program is not computing the average correctly. By looking at the program listing, you can see that the error occurs because the loop counter (N) is incremented a final time when you enter 9999 to terminate entering numbers. The value n must be decremented by 1.

To correct the error, edit the following line in the source file:

```
40   AVERAG = TOTAL/N
```

The corrected line follows:

```
AVERAG = TOTAL/(N-1)
```

Now, repeat the compile, link, and run steps:

```
$ FORTRAN average
$ LINK average
$ RUN average
ENTER NUMBER, END WITH 9999
33
ENTER NUMBER, END WITH 9999
66
ENTER NUMBER; END WITH 9999
99
ENTER NUMBER, END WITH 9999
9999
AVERAGE IS   66.00
FORTRAN STOP
$
```

In this example, the bug was easy to spot; this is not usually the case, however, and you may need to investigate a program further to debug it.

4.5.2.5 Debugging the Program

The VAX/VMS operating system has a **debugger**, which is a program that permits you to find and correct errors in your programs interactively. When you want to use the debugger, you must first compile the source program with the /DEBUG and /NOOPTIMIZE qualifiers, as follows:

```
$ FORTRAN/DEBUG/NOOPTIMIZE average
```

The /NOOPTIMIZE qualifier prevents the debugger from rearranging your source code.

When the compilation completes, use the /DEBUG qualifier when you link the object module:

```
$ LINK/DEBUG average
```

Now, when you use the RUN command to execute the program image AVERAGE.EXE, the debugger takes control, and you can use debugging commands to stop the execution of the program at a particular statement and examine or modify a variable.

For online information about the debugger, enter the HELP command at the debugger prompt as follows:

```
DBG>HELP
```

For detailed information about the debugger, see the *VAX/VMS Utilities Reference Volume.*

4.5.3 A MACRO Program

The steps required to prepare a VAX MACRO program to run with VAX/VMS are illustrated in Figure 4–4. Figure 4–4 also notes the default file types used by the MACRO, LINK, and RUN commands. For any of these commands, you can specify an explicit file type to override the default when you name an input or output file.

Note: The VAX MACRO assembler is referred to simply as MACRO throughout this manual.

4.5.3.1 Creating the Source Program

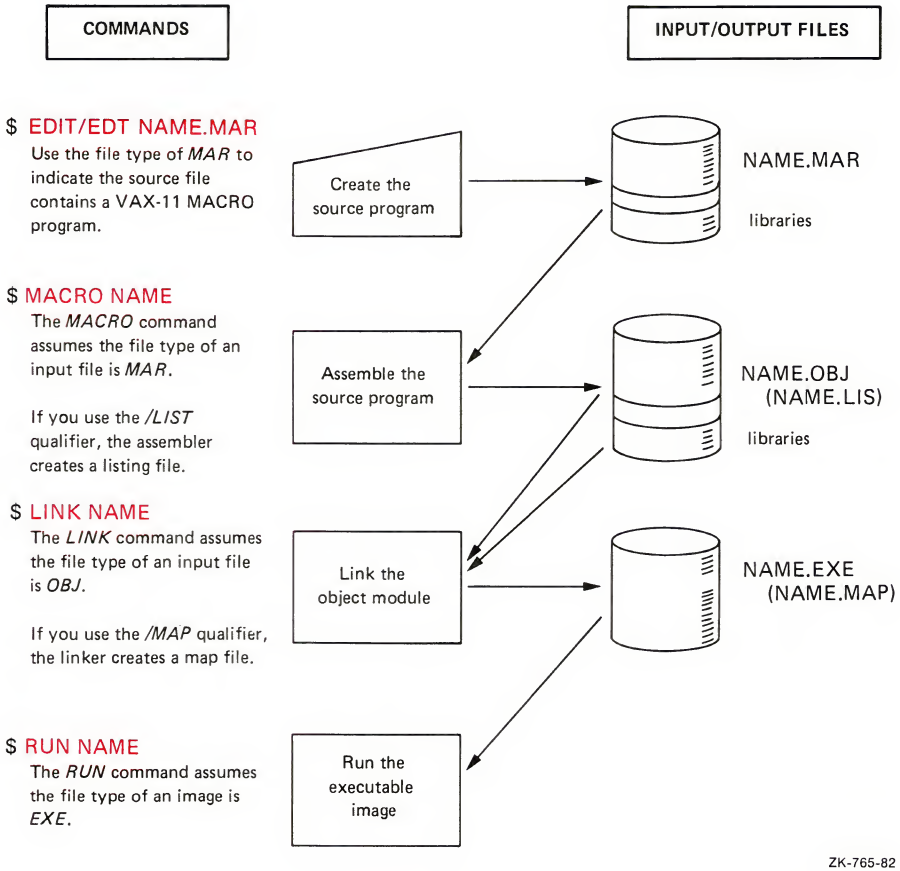
Use an editor to create a source program interactively. For example, to create the MACRO program called NAME, you can issue the DCL command EDIT. The EDIT command invokes EDT, which is the default editor for VAX/VMS:

```
$ EDIT name.mar
```

The program NAME is shown below. When you type the input statements, you can use the TAB key to align the operand and comments columns.

The program uses VAX RMS to read and write lines to the current terminal; it issues a prompting message asking for the user's name and redisplays whatever is entered in response. This program purposely has a syntax error and a bug, so you can get an idea of how to use VAX/VMS to correct programming errors.

Figure 4–4 Commands for MACRO Program Development



```
        .TITLE NAME
        .IDENT /01/
        .PSECT  RWDATA,WRT,NOEXE
; DEFINE CONTROL BLOCKS FOR TERMINAL INPUT AND OUTPUT
TRMFAB: $FAB    FNM=TT:,RAT=CR,FAC=<GET,PUT> ;FAB FOR TERMINAL
TRMRAB: $RAB    FAB=TRMFAB,UBF=BUFFER,USZ=BUFSIZ, -
                ROP=PMT, PBF=PMSG1, PSZ=P1SIZ
BUFFER: .BLKB   132                     ; INPUT READ BUFFER
BUFSIZ= .-BUFFER                        ; BUFFER LENGTH
PMSG1:  .ASCII  /ENTER YOUR NAME:       ; PROMPT MESSAGE
P1SIZ=  .-PMSG1                         ; MESSAGE SIZE
OUTMSG: .ASCII  /HELLO, YOUR NAME IS/   ; OUTPUT MESSAGE
OUTBUF: .BLKB   30                      ; MOVE NAME HERE
OUTLEN: .LONG   OUTBUF-OUTMSG
MSGSIZ: .BLKL   1                       ; ADD LENGTHS HERE
        .PSECT  NAME,EXE,NOWRT
        .ENTRY  BEGIN,0                 ; ENTRY MASK
        $OPEN   FAB=TRMFAB              ; OPEN TERMINAL FILE
        BLBC    R0,ERROR                ; EXIT IF ERROR
        $CONNECT RAB=TRMRAB             ; ESTABLISH RAB
        BLBC    R0,ERROR                ; EXIT IF ERROR
        $GET    RAB=TRMRAB              ; ISSUE PROMPT
        BLBC    R0,ERROR                ; EXIT IF ERROR
; MOVE NAME ENTERED INTO OUTPUT MESSAGE, AND FIX UP LENGTH
        MOVC3   TRMRAB+RAB$W_RSZ,BUFFER,OUTBUF
        MOVZWL  TRMRAB+RAB$W_RSZ,MSGSIZ
        ADDL    MSGSIZ,OUTLEN
; AFTER CONSTRUCTING OUTPUT MESSAGE, OUTPUT IT
        MOVAL   OUTMSG,TRMRAB+RAB$L_RBF ; UPDATE RAB: ADDRESS
        MOVW    MSGSIZ,TRMRAB+RAB$W_RSZ ; UPDATE RAB: SIZE
        $PUT    RAB=TRMRAB
        BLBC    R0,ERROR                ; EXIT IF ERROR
; ALL DONE, CLOSE THE FILE
        $CLOSE  FAB=TRMFAB
ERROR:
        RET
        .END    BEGIN
```

4.5.3.2 The MACRO Command

When you enter the MACRO command, the MACRO assembler does the following by default:

1. Produces an object module that has the same file name as the source file and a file type of OBJ
2. Uses MACRO assembler defaults when it creates output files (qualifiers on the command string can override these defaults)
3. Searches the system macro library for definitions for system macros, such as the VAX RMS macros $FAB and $RAB used in the sample program NAME.MAR

To assemble the source program NAME, issue the following command:

```
$ MACRO/LIST name
```

Since the MACRO command assumes a file type of MAR, you need not specify the file type when you name the file to be assembled. The /LIST qualifier indicates that you want a listing of the program; if there are any errors in the assembly, you may need the listing to determine what the errors are.

If the assembly is successful—that is, if the assembler did not detect any errors—the system displays a prompt for the next command:

```
$
```

If errors occur, a message is displayed at the terminal. If you entered the source program NAME exactly as it appeared above, then you received the following error message:

```
     45 47 41 53 53 45 40 20 54 50  0130
%MACRO-E-UNTERMARG, Unterminated argument
There were 1 error, 0 warnings, and 0 information messages on lines:
15(1)
MACRO/LIST NAME
```

This message indicates that the ASCII string argument coded on line 15 is incorrect; you must terminate the string with a slash (/) character.

To correct the error, edit the following line in the source file:

```
PMSG1:  .ASCII   /ENTER YOUR NAME:      ; PROMPT MESSAGE
```

The corrected line, which contains the slash character, follows:

```
PMSG1:  .ASCII   /ENTER YOUR NAME:/     ; PROMPT MESSAGE
```

Now, you can reassemble the program by entering the following command string:

```
$ MACRO/LIST name
```

If the program assembles successfully this time, you can go on to the next step. Otherwise, repeat the procedure of looking at the listing, correcting the source file, and assembling it.

4.5.3.3 Linking the Object Module

To link the program NAME, issue the LINK command as follows:

```
$ LINK name
```

This LINK command creates a file named NAME.EXE, which is an executable program image. The linker automatically includes in the executable image any library procedures required by the VAX RMS routines used.

4.5.3.4 Running the Program

To execute the program NAME, use the RUN command. When you issue the RUN command, you provide the name of an executable image. By default, the RUN command assumes a file type of EXE. Thus, to run the program NAME, type the RUN command as follows:

```
$ RUN name
```

NAME is interactive: it prompts you to enter your name, then it creates an output string from the string you entered and outputs it. A typical run of this program might appear as follows:

```
ENTER YOUR NAME: YORICK
HELLO,
$
```

As you can see, the program is writing only the first 6 characters of the output message. If you examine the listing, you can see that on line 43 the MOVW instruction places the wrong length in the buffer size field of the RAB; it uses the MSGSIZ field (that is, the length of the string you entered) rather than the sum of the string you entered and the OUTMSG string.

To correct the error, edit the source file again:

```
$ EDIT name.mar
```

Edit the following line in the source file:

```
MOVW     MSGSIZ, TRMRAB+RAB$W_RSZ;  UPDATE RAB: SIZE
```

The corrected line follows:

```
MOVW     OUTLEN, TRMRAB+RAB$W_RSZ;  UPDATE RAB:SIZE
```

Now, repeat the assembling, linking, and running:

```
$ MACRO name
$ LINK name
$ RUN name
ENTER YOUR NAME: YORICK
HELLO, YOUR NAME IS YORICK
$
```

In this example, the bug was easy to spot; this is not always the case, however, and you may need to investigate a program further to debug it.

4.5.3.5 Debugging the Program

The VAX/VMS operating system has a debugger, a program that permits you to debug your programs interactively. When you want to use the debugger, you can assemble the source program with the /ENABLE=DEBUG qualifier, as follows:

```
$ MACRO/ENABLE=debug name
```

This qualifier requests the assembler to include, in the object module, special information the debugger can use. When you link the object module you must specify the /DEBUG qualifier to link the debugger program with your program, as the following command string shows:

```
$ LINK/DEBUG name
```

Now when you use the RUN command to execute the program image NAME.EXE, the debugger takes control, and you can use debugging commands to stop the execution of the program at a particular instruction and examine or modify a variable.

For information on how to use the debugger, see the *VAX/VMS Utilities Reference Volume.*

4.6 Using Logical Names for Programming Needs

When you design programs to read and write data, you can code the programs to read or write different files each time you run them. This is called device and file independence. In the VAX/VMS operating system, device independence is accomplished through the use of logical names.

This section discusses how to make a program more efficient by using logical names.

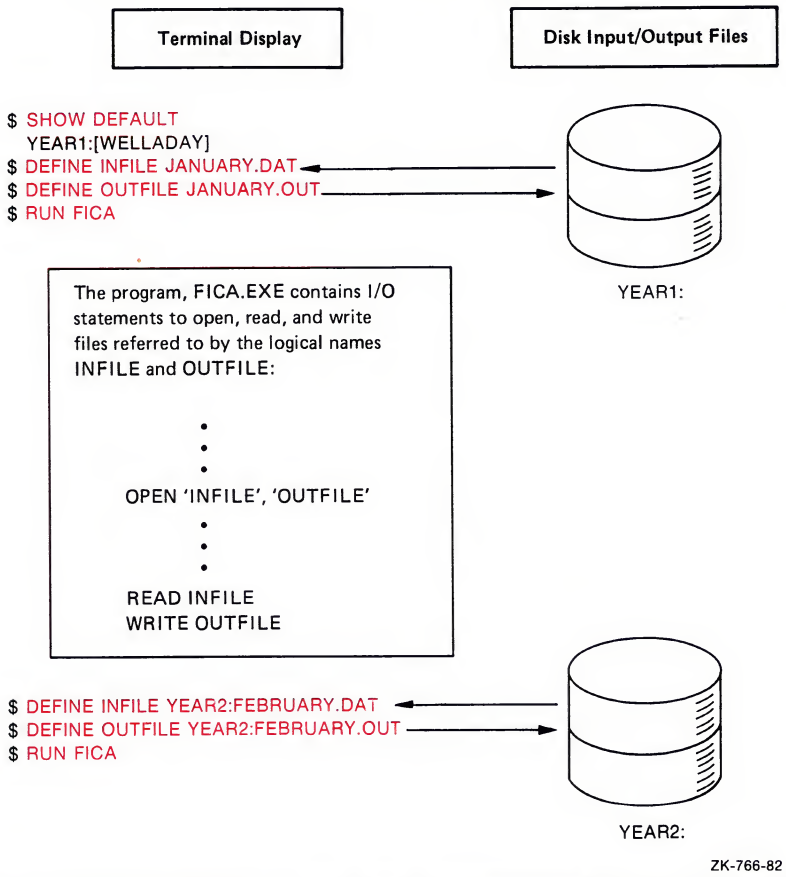
4.6.1 Using Logical Names in Programs

When you code a program, you refer to an input or output file according to the syntax requirements of the language you are using. After the program is compiled and linked, but before you run it, you can use the DEFINE command to make a connection between the logical names you used in the program and the actual files or devices you want to use when you run the program.

Figure 4–5 shows how logical names are used. The program FICA contains OPEN, READ, and WRITE statements in a general form; the program reads from a file referred to by the logical name INFILE, and writes to a file referred to by the logical name OUTFILE.

For different runs of the program, the DEFINE command establishes different equivalence names for INFILE and OUTFILE. In the first example, the program reads the file JANUARY.DAT from the device YEAR1: and writes to the file JANUARY.OUT on the same device. In the second example, it reads the file FEBRUARY.DAT from the device YEAR2: and writes the file FEBRUARY.OUT to that device.

Figure 4–5 Using Logical Names



4.7 For More Information

The three program examples presented in this chapter show only the simplest cases, using defaults for getting a program to run. VAX/VMS provides many capabilities beyond those presented in these examples. Some of the VAX/VMS manuals you may find useful are described below.

- The *VAX/VMS DCL Dictionary* contains reference information for all the commands that have been used in the examples in this chapter.

- The *Guide to Using DCL and Command Procedures on VAX/VMS* supplies more information about logical names.
- For information about a high-level language, see the document set for that language.

  Separate documentation exists for the following VAX languages:

  VAX BASIC
  VAX BLISS
  VAX C
  VAX COBOL
  VAX CORAL
  VAX FORTRAN
  VAX PASCAL
  VAX PL/I

5 Using Symbols and Command Procedures to Save Time

This chapter provides some elementary information on how you can adapt the command language to your individual needs. For example, you can:

- Establish synonyms to use in place of command names and entire command strings
- Establish default qualifiers for commands
- Create command procedures to perform a series of DCL commands
- Submit command procedures as batch jobs

You can simplify the command language to save yourself time during interactive terminal sessions. You can establish your own default commands and command qualifiers and can also create **command procedures**. These command procedures will enable you to execute a series of DCL commands by entering one command.

5.1 Abbreviating DCL Commands with Symbols

When you find yourself using long DCL commands on a regular basis, you can save time by equating them to symbols. For example, you can equate the symbol ST to the DCL command SHOW TIME:

```
$ ST = "SHOW TIME"
```

After you equate a symbol to an expression, the symbol has a new identity or value. In the previous example, the symbol ST has a new identity as the DCL command SHOW TIME. Now, you can use the symbol ST in place of SHOW TIME:

```
$ ST
    9-JUL-1986  10:45:19
```

The three parts of a symbol equation follow:

1. The symbol (for example, ST)
2. An equal sign (=)
3. The expression (for example, "SHOW TIME")

You need to supply a symbol. A symbol can be any alphanumeric string. Follow the symbol with an equal sign (=) that will assign the value of the expression to the symbol. You also need to supply an expression. An expression can be either a character string (for example, SHOW PROCESS) or an arithmetic string (for example, 8). Enclose the expression in quotation marks (") if it is a character string.

You can simplify the use of the DCL command SHOW USERS by equating it to a symbol, for example LOOK, as follows:

```
$ LOOK = "SHOW USERS"
```

Instead of typing the DCL command SHOW USERS, you can now type the word LOOK for a listing of all the current users on the system. In this example, LOOK is the symbol and SHOW USERS is the expression.

You can use the DCL command SHOW SYMBOL to see the value of any symbol you create. To see the value of the symbol LOOK, enter the following command string:

```
$ SHOW SYMBOL LOOK
 LOOK = "SHOW USERS"
$
```

You can also equate a symbol to an arithmetic string. You do not need to enclose an arithmetic string in quotation marks ("). (For detailed information about syntax and grammar rules, see the *VAX/VMS DCL Dictionary*.

The following example shows how to equate the symbol WEIGHT to the arithmetic expression 125:

```
$ WEIGHT = 125
```

Now, the symbol WEIGHT has the value of 125. You can substitute the symbol WEIGHT for the number 125. To see the value of the symbol WEIGHT, enter the following command string:

```
$ SHOW SYMBOL WEIGHT
 WEIGHT = 125    Hex = 0000007D  Octal = 00000000175
```

Notice that the system displays these numbers in three forms: decimal, hexadecimal, and octal.

You can use symbols to collect the results of arithmetic strings. For example, if you have a string of numbers that you want to add, you can equate the sum to a symbol, like TOTAL. The following example equates the symbol TOTAL to the sum of the mathematical string, 2 + 3 + 4 + 5:

```
$ TOTAL = 2 + 3 + 4 + 5
```

To see the value of the symbol TOTAL, enter the following command:

```
$ SHOW SYMBOL TOTAL
 TOTAL = 14    Hex = 0000000E  Octal = 00000000016
```

You can combine character strings within expressions. For example, to add the character string "buda" to the character string "pest" , you enter the following command string:

```
$ CITY = "Buda" + "pest"
```

To see the value of the symbol CITY, enter the following command string:

```
$ SHOW SYMBOL CITY
 CITY = "Budapest"
```

You can add many character strings together, as follows:

```
$ SONG = "One " + "pill " + "makes " + "you " + "larger"
```

To see the value of the symbol SONG, enter the following command string:

```
$ SHOW SYMBOL SONG
 SONG = "One pill makes you larger"
```

You can save time by equating long command strings to symbols. For example, for a user named BERGMAN to move from a subdirectory to his main directory, he would enter the following command string:

```
$ SET DEFAULT WORK6:[BERGMAN]
```

To save time, BERGMAN can equate this command string to a symbol (HOME), as follows:

```
$ HOME = "SET DEFAULT WORK6:[BERGMAN]"
```

Now, to move to his main directory, the user BERGMAN only needs to type the symbol HOME, as follows:

```
$ HOME
```

Symbols can be defined for command strings containing qualifiers as well as for the command itself. For example, if you want to define a synonym for the DIRECTORY command that automatically includes the /FULL qualifier, you can define the symbol LIST as follows:

```
$ LIST = "DIRECTORY/FULL"
```

Then, if you issue the following command string, the system substitutes the command DIRECTORY/FULL for the symbol LIST:

```
$ LIST myfile.dat
```

The system executes the command string DIRECTORY/FULL MYFILE.DAT, displaying detailed information about the files in your directory.

Sometimes you must place apostrophes around a symbol to identify it as a symbol to the system. Then, when the system sees the apostrophes, it knows to perform symbol substitution. For example, you can equate a symbol to a long file name:

```
$ best = "3145chapter_on_songs_of_1990.dat"
```

To type the file, all you need to enter is the TYPE command with the symbol name enclosed in apostrophes (′):

```
$ TYPE 'best'
```

See the *VAX/VMS DCL Dictionary* and the *Guide to Using DCL and Command Procedures on VAX/VMS* for detailed information about the usage of apostrophes with symbols.

Symbols can be concatenated with other symbols or items on a command string. In this case, you must enclose the symbol in apostrophes (') to indicate to the system that it must perform symbol substitution. For example, you can assign the symbol PQUALS to the following qualifiers for the PRINT command:

```
$ pquals = "/copies=2/forms=4/noburst"
```

Then, to use the symbol with the PRINT command, you must enclose it in apostrophes:

```
$ PRINT report.dat'pquals'
```

The system recognizes the apostrophes and substitutes the appropriate value (in this case the following string of qualifiers) for the symbol PQUALS:

```
/COPIES=2/FORMS=4/NOBURST
```

For more information about the effect of these qualifiers on the PRINT command, as well as rules about when to use apostrophes for symbol substitution, see the *VAX/VMS DCL Dictionary*.

Note that any symbols you assign will disappear when you log out unless you put them in a LOGIN.COM file. Section 5.2.5 explains how to create and use a LOGIN.COM file.

5.2 Creating and Executing a Command Procedure

A command procedure is a file that contains a sequence of DCL commands. You create a command procedure by using a text editor (like EDT) to create a file. Then you fill the file with DCL commands. When you invoke the command procedure, the commands are executed beginning with the first command and continuing consecutively to the end of the procedure. Each command is executed as if you had typed it.

The default file type for a command procedure file is COM.

The following example shows how to create a command procedure named CLEAN.COM that will purge your default directory and then display the remaining files in that directory. (The exclamation mark introduces a comment—text that is ignored when the command procedure is executed.)

Use a text editor to create a file named CLEAN.COM. Copy the following three lines of text into the file:

```
$ ! Purge files and look
$ PURGE
$ DIRECTORY
```

Exit from the editor. Execute the command procedure by typing an at sign (@) followed by the name of the file containing the commands. For example, to execute the command procedure CLEAN.COM, enter the following command string:

```
$ @CLEAN
```

You can invoke a command procedure from any directory. But, if the command procedure is not located in your current directory, you must precede the command procedure name by the name of the directory in which it is located. For example, if your current directory is [BASIL.FOREIGN], but the command procedure you want to invoke (CLEAN.COM) is in a different directory named [BASIL.DOMESTIC], then you would enter the following command string to invoke CLEAN.COM:

```
$ @[BASIL.DOMESTIC]CLEAN.COM
```

The rules for formatting a command procedure are:

- Begin each command string with a dollar sign—If a command in your command procedure requires information that you would normally type in, put that information on lines without dollar signs (data lines) following the command. For example, to use MAIL from a command procedure, put your responses to the prompts on data lines following the MAIL command.
- Do not abbreviate commands—Although abbreviation is allowed, the command procedure is easier to read if all of the commands are spelled out.
- Begin comments with an exclamation point—Comments explain what the procedure is doing; they are especially helpful in complex command procedures.

The following command procedure (SEND.COM) sends a message to node EBONY, with a note to the accounting department. (The CTRL/Z that usually ends the message is not necessary because the end of the command procedure indicates the end of the message):

```
                +-----------------SEND.COM----------------+
      comment ->| $ ! Send a message to EBONY              |
command string ->| $ MAIL                                  |
            +---| SEND                                    |
            |   | EBONY::USER                             |
data lines ->|   | Attention Accounting                    |
            |   | Please forward the RAZORON transactions. |
            +---| Thanks.                                 |
                +-----------------------------------------+
```

Do not put comments on data lines. If you do, DCL will treat the comments as data when it reads the information from the data lines. To change the information on the data lines, you must edit the command procedure.

5.2.1 Passing Information

When you want to pass information to a command procedure, you can have the procedure request a value for a symbol. You type in a value and your command procedure uses the symbol equated to that value in subsequent commands. When you want your command procedure to pass information back to you, you can have the command procedure display information to the terminal.

5.2.1.1 Requesting Information with the INQUIRE Command

If your command procedure needs information from you, use the INQUIRE command to request it. The INQUIRE command prompts you for information, then puts your response into a symbol. This command requires two parameters: the first is the symbol name and the second is the prompt. Use descriptive prompts (for example, "Enter a file name") to keep your command procedures clear.

The following example shows how the INQUIRE command displays the prompt "File: " and puts the user's response in the symbol OLD_FILE. (DCL automatically adds the colon and the space to the prompt that you specify.) The command procedure uses OLD_FILE in the PRINT and PURGE commands. (See the *VAX/VMS DCL Dictionary* and the *Guide to Using DCL and Command Procedures on VAX/VMS* for detailed information about using apostrophes.)

Use an editor to create a file named NEW_CLEAN.COM. Copy the following four lines of text into the file:

```
$ ! Print all versions and purge
$ INQUIRE OLD_FILE "File"
$ PRINT 'OLD_FILE';*
$ PURGE 'OLD_FILE'
```

To execute the command procedure, enter the following command string:

```
$ @NEW_CLEAN
```

You will be prompted for a file name as the following example shows:

```
File: BILLS.DAT
```

To make your command procedures more flexible, you can write them so that they require parameters. In this case, you include the requested information in the command string. No prompts are given. In order to use a command procedure that requires parameters, you must already know what parameters the procedure expects. (Parameters to a command procedure must be separated from the procedure name and from each other by one or more tabs or spaces.)

The system assigns the parameter values to special symbols named P1, P2, P3, and so on through P8. These symbols can be referenced only from within the command procedure. If you pass a command procedure more than eight parameters, the system displays an error message and does not execute the procedure.

The following command procedure contains two special symbols, P1 and P2:

```
+------NEWER_CLEAN.COM--------+
| $ ! Print and purge a file  |
| $ ! P1 and P2 are file names|
| $ PRINT 'P1';*              |
| $ PURGE 'P1'                |
| $ PRINT 'P2';*              |
| $ PURGE 'P2'                |
+-----------------------------+
```

The following command string equates P1 to BILLS.DAT and P2 to RECEIPTS.DAT:

```
$ @NEWER_CLEAN BILLS.DAT RECEIPTS.DAT
```

When this command string invokes the command procedure, the files BILLS.DAT and RECEIPTS.DAT are printed and purged. Note that the parameters are explained in comments at the beginning of the procedure. These comments make the command procedure easier to read and understand.

5.2.1.2 Displaying Information with the WRITE Command

When you want your command procedure to display information on your screen, use the WRITE command. The WRITE command takes two parameters: the first parameter tells DCL where to display the text; the second parameter tells DCL what text to display. To tell DCL that you want the text displayed on the terminal, use the logical name SYS$OUTPUT as the first parameter. (When you log in, the system automatically equates SYS$OUTPUT to the output stream for your terminal. The system uses this output stream for prompting and informational messages.)

If you want to display a line of text, enclose the text in quotation marks and include it as the second parameter. For example, create a file named FUN.COM and fill it with the following line:

```
+----------------------FUN.COM---------------------------+
|                                                        |
| $ WRITE SYS$OUTPUT "Hello Dolly"                       |
|                                                        |
+--------------------------------------------------------+
```

Now, enter the command @FUN to see the text "Hello Dolly"

The following command procedure displays the text "All versions printed; file purged."

```
+-------------------------CLEAN.COM------------------------+
| $ PRINT 'P1';*                                           |
| $ PURGE 'P1'                                             |
| $ WRITE SYS$OUTPUT "All versions printed; file purged."  |
+----------------------------------------------------------+
```

When you want to display the value of a symbol, include the symbol as the second parameter. For example, the third line of the following command procedure (CLEAN1.COM) contains the WRITE command followed by the first parameter (SYS$OUTPUT) and the second parameter (P1):

```
+------------------------CLEAN1.COM-----------------------+
| $ PRINT 'P1';*                                          |
| $ PURGE 'P1'                                            |
| $ WRITE SYS$OUTPUT P1                                   |
+---------------------------------------------------------+
```

After you enter the following command string, the command procedure CLEAN1.COM displays the value of P1 as BILLS.DAT:

```
$ @CLEAN BILLS.DAT
```

If you want to display a line of text and the value of one or more symbols, include the symbols in the text and specify the entire line as the second parameter. Each symbol in the text must be preceded by two apostrophes and followed by one apostrophe. For example, when the following command procedure is invoked with the command string @CLEAN BILLS.DAT, the procedure displays the message "All versions of BILLS.DAT were printed; file was purged."

```
+-------------------------CLEAN.COM-----------------------+
| $ PRINT 'P1';*                                          |
| $ PURGE 'P1'                                            |
| $ WRITE SYS$OUTPUT -                                    |
| "All versions of ''P1' were printed; file was purged."  |
+---------------------------------------------------------+
```

5.2.2 Editing a File with the EDIT Command

If you want your command procedure to allow you to edit a file, use the EDIT command. EDIT reads information from the location equated to the logical name SYS$INPUT. When you log in, the system automatically equates SYS$INPUT to the input stream for your terminal. (The system uses this input stream to read commands and data that are required by commands.) However, when you invoke a command procedure, the system changes SYS$INPUT, equating it to the command procedure. In order for EDIT to read information from the terminal, you must use the following command to equate SYS$INPUT to the terminal again:

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
```

The /USER_MODE qualifier causes the new logical name definition to remain in effect only for the command immediately following. In general, you should use this qualifier when you are changing system defined logical names.

The following command procedure asks you for a file name, then invokes the editor so that you can edit the file (this type of command sequence would generally be part of a larger command procedure):

```
+----------------------------------------+
|                   .                    |
|                   .                    |
|                   .                    |
| ! EDIT file of user's choice           |
| $ INQUIRE FILE_NAME "File"             |
| $ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND|
| $ EDIT 'FILE_NAME'                     |
|                   .                    |
|                   .                    |
|                   .                    |
+----------------------------------------+
```

5.2.3 Using Logic

When you want your command procedure to choose a course of action depending on some piece of information you provide, use the IF command. When you want your command procedure to skip a number of steps for some reason, use the GOTO command.

The IF command conditionally executes a command. You specify both the condition and the command in the following format:

```
$ IF condition THEN command
```

When the command procedure reads an IF statement, it evaluates the condition. If the condition is true, the command executes; if it is false, the command is ignored and the system executes the statement following the IF command. A condition must be stated using the conditional operators listed in Table 5–1.

Table 5–1 Conditional Operators

Operator for Numbers	Operator for Characters	Function
.EQ.	.EQS.	Equal to
.NE.	.NES.	Not equal to
.LE.	.LES.	Less than or equal to
.LT.	.LTS.	Less than
.GE.	.GES.	Greater than or equal to
.GT.	.GTS.	Greater than

Use the GOTO command to direct the command procedure to a particular line in your command procedure. This command has the following format:

```
$ GOTO label
```

When the command procedure reads the GOTO statement, it goes to the command string in the procedure that begins with the specified label followed by a colon. Each label in a command procedure should be unique.

The following command procedure uses the IF statement to see whether you have specified a file name. If OLD_FILE is blank (""), the command GOTO ERR is executed, the command procedure continues executing at the line labeled ERR:, and displays the message "No file printed or purged." If OLD_FILE is not blank, the file named in OLD_FILE is printed and purged. When the command procedure reads the EXIT command, it exits, returning the user to the DCL command level.

```
+-------------------CLEAN.COM----------------------+
| $ ! Print all versions and purge                 |
| $ INQUIRE OLD_FILE "File"                        |
| $ IF OLD_FILE .EQS. "" THEN GOTO ERR             |
| $ PRINT 'OLD_FILE';*                             |
| $ PURGE 'OLD_FILE'                               |
| $ EXIT                                           |
| $ ERR:                                           |
| $ WRITE SYS$OUTPUT "No files printed or purged." |
| $ EXIT                                           |
+--------------------------------------------------+
```

If you want to create a loop in your command procedure, perform the following steps:

1. Begin the loop with a label.
2. Write the body of the loop. Within the body of the loop you must have an IF statement that directs the command procedure to go to the label that marks the end of the loop when the condition is true. If you do not include this statement or you write the procedure so that the condition is never true, the command procedure remains in an "infinite loop" (in which case, you must press CTRL/Y to abort the procedure and return to DCL command level).
3. Go back to the label that marks the beginning of the loop and end the loop with a label.

The following command procedure uses a loop to add a stream of numbers that you enter. When you enter a zero, the command procedure exits from the loop. (Indenting the commands two spaces makes the command procedure easier to read. It does not affect the way it executes.)

```
+------------------------ADD.COM-----------------------+
| $ ! Addition program                                 |
| $                                                    |
| $ ! initialize the sum                               |
| $   SUM = 0                                          |
| $                                                    |
| $ ! begin loop                                       |
| $  LOOP:                                             |
| $ ! get a number from the user                       |
| $   INQUIRE NUM "Enter the first number (0 to end)"  |
| $ ! if NUM is 0, exit                                |
| $   IF NUM .EQ. 0 THEN GOTO DONE                     |
| $ ! add the number to the current total              |
| $   SUM = SUM + NUM                                  |
| $ ! go to the beginning of the loop                  |
| $   GOTO LOOP                                        |
| $                                                    |
| $ ! write the sum before exiting                     |
| $ DONE:                                              |
| $   WRITE SYS$OUTPUT "The sum of the numbers is ",SUM |
| $   EXIT                                             |
+------------------------------------------------------+
```

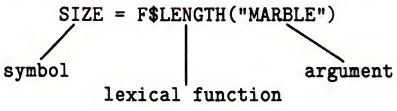
5.2.4 Extracting Information with Lexical Functions

A **lexical function** is like a symbol in that it is equated to a value. However, a lexical function obtains its value by reading arguments supplied by you and by performing a particular operation. You can think of lexical functions as helpers. Each lexical function performs its own special task. For example, F$LENGTH reads a character string that you supply and then replaces itself with the length of that character string.

Enter the following command string to make F$LENGTH calculate the length of the string MARBLE. The result of the calculation is put in the symbol SIZE.

```
$ SIZE = F$LENGTH("MARBLE")
```

Labels for the various parts of the command string follow:



To see the result of the calculation (the value of the symbol SIZE), enter the following command string:

```
$ SHOW SYMBOL SIZE
      SIZE = 6    Hex = 00000006    Octal = 00000000006
```

Instead of supplying a specific value for the argument for a lexical function, you can use a symbol. Then, you can equate varying values to the symbol. For example, enter the following command string to equate the symbol ANYTHING to the string "BLAST" :

```
$ ANYTHING = "BLAST"
```

Enter the following command string to make F$LENGTH calculate the length of the value of the symbol ANYTHING. (In this case, the symbol ANYTHING is equated to the string BLAST):

```
$ SIZE = F$LENGTH(ANYTHING)
```

In the previous example, the symbol ANYTHING equates to the string "BLAST" . The lexical function F$LENGTH determines the length of "BLAST" and equates this length to the symbol SIZE.

To see the result of the calculation (the value of the symbol SIZE), enter the following command string:

```
$ SHOW SYMBOL SIZE
      SIZE = 5    Hex = 00000005    Octal = 00000000005
```

When the length of the argument changes, F$LENGTH returns a different number, as the following example shows:

```
$ ANYTHING = "TEA"
$ SIZE = F$LENGTH (ANYTHING)
$ SHOW SYMBOL SIZE
      SIZE = 3    Hex = 00000003    Octal = 00000000003
```

In the previous example, the symbol ANYTHING equates to the string "TEA" . The lexical function F$LENGTH determines the length of "TEA" and equates the symbol SIZE to this length.

Table 5–2 lists some commonly used lexical functions with their required arguments.

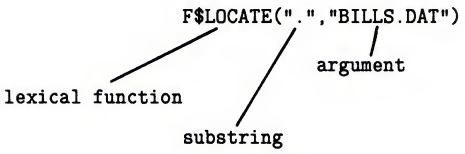
Table 5–2 Common Lexical Functions

Function	Description
F$EXTRACT(offset,length,string)	Returns a substring
F$LENGTH(string)	Returns the length of a character string
F$LOCATE(substring,string)	Returns the location of a substring
F$MODE( )	Returns a character string showing the mode in which a process is executing
F$STRING(expression)	Converts a number to a character string
F$TIME( )	Returns the current date and time

For detailed information about lexical functions and their required arguments, see the *VAX/VMS DCL Dictionary*.

In the following command procedure, F$LOCATE and F$LENGTH are used to determine whether you have included a file type with the file name in OLD_FILE. (A period indicates that a file type has been included in the file name). The lexical function F$LOCATE searches a string for a substring

and returns the position of the substring in the string. For instance, the lexical function F$LOCATE(^." ,^"BILLS.DAT") is replaced by the number six because the substring (.) is in the sixth position of the string BILLS.DAT:



If the substring is not found (in this case, a period ^."), the value of F$LOCATE is the length of the string being searched. The following command procedure compares F$LOCATE (the location of the substring ^.") to F$LENGTH (the length of the string) to see whether or not there is a period in OLD_FILE:

```
$ ! Check for file type in the name (just the .)
$ LOOP:
$ INQUIRE OLD_FILE "File"
$ IF F$LOCATE(".",OLD_FILE) .NE. F$LENGTH(OLD_FILE) -
THEN GOTO END_LOOP
$
$ ! Write error message and repeat loop
$ WRITE SYS$OUTPUT "Include the file type in the file name"
$ GOTO LOOP
$
$ END_LOOP:
$ WRITE SYS$OUTPUT "Yes, you have supplied a file type"
```

If you supply a file name with a file type, the command procedure will display the following text and return you to the DCL command level:

```
Yes, you have supplied a file type
```

If you supply a file name without a file type, you will be prompted to reenter the file name with a file type.

5.2.5 What is a LOGIN.COM File?

If you become a frequent user of the VAX/VMS system, you may find that you are entering the same sequence of commands or assignment statements every time you log in. To avoid such repetition, you can place these commands and statements in a special command procedure.

You must name the file containing the command procedure LOGIN.COM and create it in your default disk directory. Like other command procedures, your LOGIN.COM file can be created by an editor, like EDT. When you log in to the system, the system automatically searches for a file with this file name. If the system locates the LOGIN.COM file, the system automatically executes the commands within that file.

A LOGIN.COM file might contain the following definitions:

Figure 5–1 Looking at a LOGIN.COM File

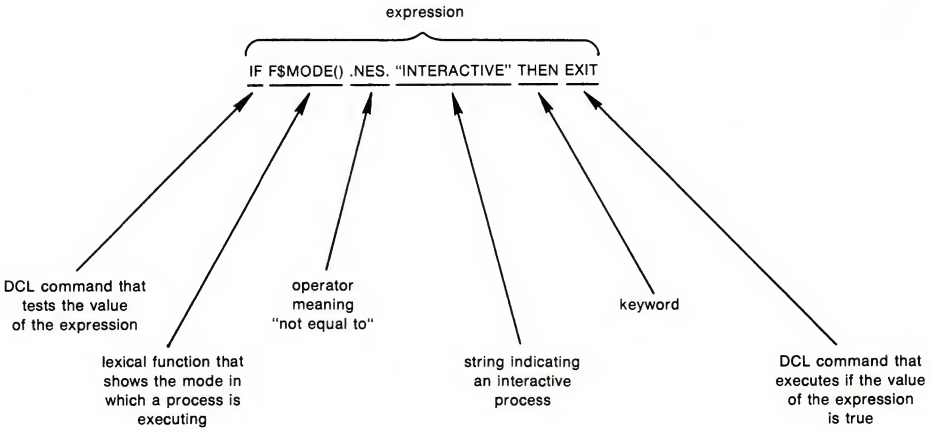
```
$IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
$!symbols for commands I frequently use
$ST=="SHOW TIME"
$SHQU=="SHOW QUEUE"
$!symbols for my directories
$HOME=="SET DEFAULT WORK6:[MALCOLM]"
$TEST=="SET DEFAULT [MALCOLM.TESTFILES]"
$BILLS=="SET DEFAULT [MALCOLM.BILLS]"
$!logical names for people to whom I send mail
$DEFINE AL ALBINONI
$DEFINE BRO BREATH::BROKOWITZ
$DEFINE SAL TITIAN::SALIMONI
```

Note that all the symbols defined in the LOGIN.COM file in Figure 5–1 are global symbols, assigned with two equal signs. If these symbols were local (assigned with one equal sign) they would be recognized only within the LOGIN.COM file, and would therefore be useless to you.

Also note the first line in the LOGIN.COM file in Figure 5–1. This line translates into the following:

```
"If the current process is not interactive, then ignore
the rest of this LOGIN.COM file".
```

Figure 5–2 shows the labeled line.



You must include this line in your LOGIN.COM file because your LOGIN.COM file may contain commands that are specific to an interactive environment that would cause a batch or network job to abort. (For information about batch jobs, see Section 5.2.6).

Command procedures can be executed from within other command procedures. You may want to place the global assignment statements you use for command synonyms in a separate file and execute this procedure in the LOGIN.COM file. For example, suppose the file SYNONYM.COM contains the following lines:

```
$ LIST == "DIRECTORY"
$ LO == "@LOG"
```

Your LOGIN.COM file would contain the following line:

```
$ @SYNONYM
```

The first line in the following LOGIN.COM file establishes the definitions in the file containing the command procedure SYNONYM.COM:

```
$!symbols for commands I frequently use
$ @SYNONYM
$ST=="SHOW TIME"
$SHQU=="SHOW QUEUE"
$!symbols for my directories
$HOME=="SET DEFAULT WORK6:[MALCOLM]"
$TEST=="SET DEFAULT [MALCOLM.TESTFILES]"
$BILLS=="SET DEFAULT [MALCOLM.BILLS]"
$!logical names for people to whom I send mail
$DEFINE AL ALBINONI
$DEFINE BRO BREATH::BROKOWITZ
$DEFINE SAL TITIAN::SALIMONI
```

5.2.6 Submitting Batch Jobs to Avoid Delays

If you have executed a command procedure, you have noticed that while the procedure is executing you cannot do anything else on your terminal. This is because your process is already executing the command procedure and it can execute only one command at a time. To avoid this delay, you can submit a command procedure as a **batch job**. A batch job is executed in a process of its own, therefore, your process is not kept waiting (however, this means that you cannot use the terminal to send information to or receive information from the command procedure while it executes).

To submit a command procedure as a batch job, use the SUBMIT command. The following example shows how to submit the command procedure CLEAN.COM in the [ACCOUNT] directory to the system as a batch job:

```
$ SUBMIT [ACCOUNT]CLEAN
```

When a command procedure is submitted as a batch job, it executes as if you had just logged in. This means that your default directory is probably your top level directory, which is not necessarily the directory that you want to use. If you want a different default directory, you must include the SET DEFAULT command in the command procedure before you reference any files.

If you are submitting a command procedure that requires parameters, you must use the /PARAMETERS qualifier with the SUBMIT command. The following command submits the command procedure CLEAN.COM in the [ACCOUNT] directory. BILLS.DAT and RECEIPTS.DAT are passed as parameters.

```
$ SUBMIT [ACCOUNT]CLEAN/PARAMETERS=(BILLS.DAT,RECEIPTS.DAT)
```

When the batch job is finished, a log file containing the output from the command procedure is printed on the line printer. After the log file is printed it is deleted. If you want to save the log file, use the /KEEP qualifier with the SUBMIT command, as shown:

```
$ SUBMIT/KEEP  [ACCOUNT]CLEAN-
$_/PARAMETERS=(BILLS.DAT,RECEIPTS.DAT)
```

When a log file is kept, it is given the same name as the command procedure, a file type of LOG, and put in your top level directory.

5.2.7 Displaying Command Lines during Execution

By default, the command lines in a command procedure are not displayed as they are executed. When you want to see each command line as it executes, use the DCL command SET VERIFY. When you create your command procedure, add the SET VERIFY command to the top of the list of commands, as follows:

```
+-------------------CLEAN.COM----------------------+
| $ ! Print all versions and purge                 |
| $ SET VERIFY                                     |
| $ INQUIRE OLD_FILE "File"                        |
| $ IF OLD_FILE .EQS. "" THEN GOTO ERR             |
| $ PRINT 'OLD_FILE';*                             |
| $ PURGE 'OLD_FILE'                               |
| $ EXIT                                           |
| $ ERR:                                           |
| $ WRITE SYS$OUTPUT "No files printed or purged." |
| $ EXIT                                           |
+--------------------------------------------------+
```

When you invoke the command procedure CLEAN, you will see every command line as it executes as follows:

```
$@ CLEAN
$ SET VERIFY
!Print all versions and purge
$ INQUIRE OLD_FILE "File"
File:memos.dat
memos.dat
$ IF OLD_FILE .EQS. ""THEN GOTO ERR
$ PRINT MEMOS.DAT
Job MEMOS (queue HAPPY_PRINT, entry 301) started on SPHERE$LPA0
$ PURGE MEMOS.DAT
$ EXIT
$
```

6 More About DCL Commands

To become familiar with the numerous DCL commands, enter the HELP HINTS command. The available DCL commands are organized by function and listed in the following categories:

- Batch_and_print_jobs
- Creating_processes
- Files_and_directories
- System_management
- Command_procedures
- Developing_programs
- Logical_names
- Terminal_environment
- Contacting_people
- Executing_programs
- Physical_devices
- User_environment

This chapter introduces some of the DCL commands in these categories. For complete information about all the available DCL commands, see the *VAX/VMS DCL Dictionary*.

6.1 Printing your Files

You need to know only one DCL command to make a hard copy of a file: PRINT. But, it is useful to know about other commands that will allow you to prevent a file from being printed after you have issued the PRINT command. The following sections describe these other commands.

6.1.1 How Do You Print a File?

To print a file, you enter the DCL command PRINT followed by the name of the file you want to print:

```
$ PRINT list.dat
```

When you enter the PRINT command, the file you specify enters a print queue. A file in the print queue is called a job and has a unique job number, called a job entry number. When you enter the PRINT command, the system responds with a message indicating the name and job entry number of the job as well as the name of the print queue. The following message indicates that a job named LIST with a job entry number of 624 was entered on a print queue named PRINT$FUN:

```
Job LIST (queue PRINT$FUN, entry 624) started on PRINT$FUN
```

For more information about the PRINT command, see Chapter 2. For detailed information about all the available qualifiers you can use with the PRINT command, see the *VAX/VMS DCL Dictionary*.

6.1.2 Looking at Jobs in the Print Queue

You can see which files are in the print queue by entering the following DCL command:

```
$ SHOW QUEUE queue-name
```

If you do not specify a queue name with the SHOW QUEUE command, you will see a list of all the available queues. The following example shows how to see the files in the print queue named PRINT$FUN:

```
$ SHOW QUEUE PRINT$FUN
Print queue PRINT$FUN
Jobname      Username      Entry      Status
-------      --------      -----      ------
LIST.DAT     BELLINI         624      Printing
```

6.1.3 Removing a Job from the Print Queue

Sometimes you may type the PRINT command, specify a file for printing, and then change your mind. When you have a job waiting in the print queue and you want to remove it before it starts printing, enter the following DCL command:

```
$ DELETE/ENTRY=job-entry-number queue-name
```

You can see the job entry number of your job by entering the SHOW QUEUE command. When you use the DELETE/ENTRY command, the system notifies you that it has deleted the job by issuing the following message:

```
Job NAMES (queue PRINT$WORK, entry 755) completed
%JBC-E-JOBDELETE, job deleted before execution
```

This message indicates that a file named NAMES.DAT with a job entry number of 755 was deleted from a print queue named PRINT$WORK before it was printed.

6.1.4 Stopping a Job that is Currently Printing

If you want to stop a file that is currently printing, enter the following DCL command:

```
$ STOP/QUEUE/ENTRY=job-entry-number queue-name
```

Again, you can see the job entry number of your job by entering the SHOW QUEUE command. The system notifies you that the job has been stopped with the following message:

```
Job ANOTHER.DAT (queue HORACE$PRINT, entry 755) completed
%JBC-E-JOBABORT, job aborted during execution
```

6.2 What is a Batch Job?

A batch job is a file containing a series of commands (and optionally input data) that is submitted to the operating system for execution. A batch job is executed in a process of its own. Therefore, batch jobs allow you to have two or more processes doing different things at the same time. For example, you may have a command procedure that you want to execute, but you also may want to use your terminal interactively. Instead of waiting for your command procedure to finish executing before doing interactive work, you can submit the command procedure as a batch job. Because a batch job is executed in a process of its own, your interactive process will not be kept waiting.

6.2.1 How Do You Start a Batch Job?

To submit a command procedure as a batch job, enter the following DCL command:

```
$ SUBMIT file-spec
```

The SUBMIT command requests the operating system to place a command procedure in a batch job queue. You will see the following message:

```
Job LOOK (queue BATCH$FUN, entry 442) started on BATCH$FUN
```

If you want the system to notify you when the job is complete, use the /NOTIFY qualifier with the SUBMIT command.

While the system processes your batch job, you can continue interactive use of your terminal.

6.2.2 Looking at Jobs in the Batch Queue

To see the command procedures in the batch queue, enter the following DCL command:

```
$ SHOW QUEUE queue-name
```

The following response indicates that a command procedure named LOOK.COM with a job entry number of 442 was submitted by a user named SULLIVAN and is currently executing:

```
Jobname     Username     Entry     Status
-------     --------     -----     ------
LOOK        SULLIVAN       442     Executing
```

If you specified the /NOTIFY qualifier, you will see the following message when the batch job finishes executing:

```
Job LOOK (queue BATCH$FUN, entry 442) completed
$
```

6.2.3 Removing a Job from the Batch Queue

As with the PRINT command, you may enter the SUBMIT command, specify a command procedure for execution, and then change your mind. When you have a batch job waiting in the batch queue and you want to remove it before it starts executing, enter the following DCL command:

```
$ DELETE/ENTRY=job-entry-number queue-name
```

You can see the job entry number of your batch job by entering the SHOW QUEUE command. When you use the DELETE /ENTRY command, the system notifies you that it has deleted the batch job by issuing the following message:

```
Job LOOK (queue BATCH$FUN, entry 442) completed
%JBC-E-JOBDELETE, job deleted before execution
```

6.2.4 Stopping a Job that is Currently Executing

If you want to stop a batch job that is currently executing, enter the following DCL command:

```
$ STOP/QUEUE/ENTRY=job-entry-number queue-name
```

Again, you can see the job entry number of your batch job by entering the SHOW QUEUE command. If you specified the /NOTIFY qualifier, the system will notify you that the job has been stopped with the following message:

```
Job LOOK (queue BATCH$FUN, entry 442) completed
%JBC-E-JOBABORT, job aborted during execution
```

6.3 Sorting, Searching, Appending, Comparing, and Copying Files

The following sections introduce five DCL commands to aid you when working with files: SORT, SEARCH, APPEND, DIFFERENCES, and COPY.

6.3.1 How to Reorganize Lists

When you want to reorganize a list of items in a file, you can use the DCL command SORT. You must specify an input file to be sorted as well as a name for the newly sorted file:

```
$ SORT input-file output-file
```

The following example shows a file named SIMPLE_SORT.DAT containing a list of colors:

```
gold
brown
yellow
blue
silver
maroon
green
beige
mauve
```

When you enter the SORT command, the list is reorganized alphabetically, according to the first letter of each word.

```
$ SORT simple_sort.dat new_simple_sort.dat
```

The newly sorted file, named NEW_SIMPLE_SORT.DAT, follows:

```
beige
blue
brown
gold
green
maroon
mauve
silver
yellow
```

In a file with more than one column of items, the SORT command moves entire lines of information, not just the first column of items. But, by default, all the columns of items are sorted by the first letter of the first word in the first column. The following example shows a file named SORT_NAMES.DAT containing three columns of information (name, social security number, and profession):

Saxon, Nicholas	749-38-2317	teacher
Able, George	238-90-5674	writer
Drendon, Marka	948-50-3749	writer
Ralston, Celia	263-72-4677	dancer
Briggs, Georgia	374-83-3526	artist
Gregwitz, Marna	478-52-0026	guitarist

When you enter the SORT command, the names are sorted by the first letter of the first word:

```
$ SORT sort_names.dat new_sort_names.dat
```

The newly sorted file, NEW_SORT_NAMES.DAT, follows:

Able, George	238-90-5674	writer
Briggs, Georgia	374-83-3526	artist
Drendon, Marka	948-50-3749	writer
Gregwitz, Marna	478-52-0026	guitarist
Ralston, Celia	263-72-4677	dancer
Saxon, Nicholas	749-38-2317	teacher

By specifying qualifiers with the SORT command, you can sort information by number instead of by character. Or, you can specify that the file be sorted in descending order (ZYXWVUTSR or 87654321) instead of ascending order (ABCDEFG or 12345678). For detailed information about the numerous other ways to sort files, see the *VAX/VMS Utilities Reference Volume*.

6.3.2 Searching for a String

As you work with files, you may need to locate specific strings of text within the files. To find a specific string of text, use the DCL command SEARCH:

```
$ SEARCH file-spec "search-string"
```

You must enclose the search string with quotation marks.

For example, a file named TEST.DAT contains the following two paragraphs of text:

```
When you type commands, qualifiers, or parameters you do not
always need to type the full word.  In fact, you never have
to type more than the first four characters, and in many
cases you can type only one or two characters.  The rule to
follow is: you must type at least the minimum number of
characters necessary to make the command unique.

For example, the SET, SEARCH, and SHOW commands all begin
with the letter "S".  To make the SHOW command unique, you
must type at least two characters, SH. To make the SET and
SEARCH commands unique,  you must type three characters, SET
and SEA respectively.  The examples in this handbook show full
commands so that you can become familiar with the commands
and what they do.
```

To search for the string "characters" in the file named TEST.DAT, enter the following command string:

```
$ SEARCH test.dat "characters"
```

The SEARCH comma will display each line containing the string "characters" (lines 3,4,6,10, and 11), as follows:

```
to type more than the first four characters, and in many
cases you can type only one or two characters.  The rule to
characters necessary to make the command unique.
must type at least two characters, SH. To make the SET and
SEARCH commands unique,  you must type three characters, SET
```

For detailed information about the SEARCH command, see the *VAX/VMS DCL Dictionary*.

6.3.3 How to Append Files

When you want to add the contents of one file to the contents of another file, you can use the DCL command APPEND.

```
$ APPEND input-file-spec output-file-spec
```

The input-file-spec is the name of the file you want to add to the end of the output-file-spec. The system prompts you for the name of the input-file-spec first and then the name of the output-file-spec.

```
$  APPEND
_From: input-file-spec
_To: output-file-spec
```

The following example shows how to add the contents of a file named ARM.FUN to the end of a file named BODY.FUN.

The contents of a file named ARM.FUN follow:

```
arm arm arm arm
arm arm arm arm
arm arm arm arm
```

The contents of a file named BODY.FUN follow:

```
body body body
body body body
body body body
```

Enter the following command string:

```
$ APPEND
_From: ARM.FUN
_To: BODY.FUN
```

Now the contents of the file named ARM.FUN are appended to the end of the file named BODY.FUN, as follows:

```
body body body
body body body
body body body
arm arm arm arm
arm arm arm arm
arm arm arm arm
```

For detailed information about the qualifiers available with the APPEND command, see the *VAX/VMS DCL Dictionary*.

6.3.4 Comparing Files

To compare the contents of one file with the contents of another file, use the DCL command DIFFERENCES:

```
$ DIFFERENCES
_File 1: first input-file-spec
_File 2: second input-file-spec
```

The DIFFERENCES command lists each record in the first input file that has no match in the second input file and then lists the next record that it finds that does have a match.

The following example shows two copies of the same letter, COPY1.DAT and COPY2.DAT. There are three differences between them.

COPY1.DAT:

```
13 March 1985

Reprint Permission Department
The Doubleday Company
2709 Third Avenue
New York, New York 10022

Dear Permission Department,

I am currently writing a text processing handbook
for Digital Equipment Corporation.  This handbook
describes how to format and process text on the VAX/VMS
operating system.  I wish to make the handbook more
interesting by adding the three sentences (marked in your
copy) from page 22 of your book, "Flying without Fear", as
part of an example.

I would like to display this excerpt within an example that
explains how DIGITAL Standard Runoff fills and justifies
text.  I request permission from you to include these three
sentences in the handbook.  I will be happy to
acknowledge your permission on the copyright page of the
handbook.  If you would like any other information
about the handbook, please let me know.
```

COPY2.DAT:

```
13 March 1985

Reprint Permission Department
The Doubleday Company
2709 Third Avenue
New York, New York 10022

Dear Permissions Department,

I am currently writing a text processing handbook
for Digital Equipment Corporation.  This handbook
describes how to format and process text on the VAX/VMS
operating system.  I wish to make the handbook more
interesting by adding the three sentences (marked in your
copy) from page 32 of your book, "Flying without Fear", as
part of an example.

I would like to display this excerpt within an example that
explains how Digital Standard Runoff fills and justifies
text.  I request permission from you to include these three
sentences in the handbook.  I will be happy to
acknowledge your permission on the copyright page of the
handbook.  If you would like any other information
about the handbook, please let me know.
```

To see the differences between COPY1.DAT and COPY2.DAT, enter the following command string:

```
$ DIFFERENCES
_File 1: COPY1.DAT
_File 2: COPY2.DAT
```

The DIFFERENCES command lists the unmatching lines as it compares both files:

```
************
File DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4
   13        Dear Permission Department,
   14
******
File DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
   13        Dear Permissions Department,
   14
************
************
File DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4
   20        copy) from page 22 of your book, "Flying without Fear", as
   21        part of an example.
******
File DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
   20        copy) from page 32 of your book, "Flying without Fear", as
   21        part of an example.
************
************
File DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4
   24        explains how DIGITAL Standard Runoff fills and justifies
   25        text.  I request permission from you to include these three
******
File DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
   24        explains how Digital Standard Runoff fills and justifies
   25        text.  I request permission from you to include these three
************


Number of difference sections found: 3
Number of difference records found: 3
DIFFERENCES /IGNORE=()/MERGED=1-
    DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4-
    DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
```

For detailed information about the qualifiers available with the DIFFERENCES command, see the *VAX/VMS DCL Dictionary*.

6.3.5 How to Copy a File

To copy a file, use the COPY command. You can use it to make copies of files in your default directory, to copy files from one directory to another directory, to copy files from other devices, or to create files consisting of more than one input file.

When you enter the COPY command, you specify first the name(s) of the input file(s) you want to copy, then the name of the output file. For example, the following COPY command copies the contents of the file PAYROLL.TST to a file named PAYROLL.OLD.

```
$ COPY payroll.tst payroll.old RET
```

If a file named PAYROLL.OLD exists, a new version of that file is created with a higher version number.

To copy a file from the directory [MALCOLM] to the subdirectory [MALCOLM.TESTFILES], and give it the new name, OLDFILE.DAT, enter the following command string:

```
$ COPY newfile.dat [malcolm.testfiles]oldfile.dat RET
```

When you copy files from devices other than your default disk, you must specify the device name with the COPY command. For example, the following command string copies a file from your default directory onto an RK07 disk:

```
$ COPY payroll.tst dma1: RET
```

Note that the output file specification did not include a file name or file type; the COPY command uses the same directory, file name, and file type as the input file, by default.

Before you can copy any files to or from devices other than system disks, you must gain access to these devices. You do this by folling two steps:

1. Mounting the volume, with the MOUNT command.
2. Ensuring that the volume has a directory for cataloging the file. If no directory exists, use the CREATE command to create one.

Note that the VAX/VMS operating system protects against others accessing volumes that individuals maintain for private purposes, as well as access to system volumes. For details on the commands and procedures necessary to prepare and use disks and magnetic tapes, see the *Guide to VAX/VMS Disk and Magnetic Tape Operations* and the *VAVMS DCL Dictionary*.

6.4 Controlling Your Terminal Environment

You can use various DCL commands to modify your terminal environment. This section introduces several of these commands:

- SET PROMPT
- SET TERMINAL
- SHOW TERMINAL
- RECALL (CTRL/B)
- DEFINE/KEY
- SHOW KEY
- DELETE/KEY

6.4.1 Changing Your System Prompt

The first way you may want to alter your terminal environment is to change your system prompt. By default, VAX/VMS displays a dollar sign ($) when you are at the DCL command level. To specify a different prompt for the DCL command level, use the following command:

```
$ SET PROMPT = prompt-string
```

For example, to change your prompt to GOOD_MORNING>, enter the following command string:

```
$ SET PROMPT = good_morning>
GOOD_MORNING>
```

When you want your prompt-string to retain lowercase letters, enclose the string in quotation marks (″). Otherwise, letters are automatically converted to uppercase. For example, enter the following command string:

```
$ SET PROMPT = "good morning>"
good morning>
```

When you log out and back in again, the default DCL command prompt, the dollar sign, is restored.

To save a prompt from session to session, enter the SET PROMPT command in your LOGIN.COM file, as follows:

```
$SET PROMPT ="SMILE>"
```

The prompt SMILE> will remain in effect until you change or delete the command line in your LOGIN.COM file.

For more information about the SET PROMPT command, see the *VAX/VMS DCL Dictionary*.

6.4.2 Changing Your Terminal Characteristics

To see the characteristics set for your terminal, enter the DCL command SHOW TERMINAL. To change your terminal characteristics, you can specify various qualifiers with the DCL command SET TERMINAL. Each characteristic is controlled by a qualifier. For detailed information about all the available qualifiers, see the *VAX/VMS DCL Dictionary*.

6.4.3 Displaying Previously Entered Commands

During an interactive session, you may enter many command strings. To see these command strings after you have processed them, enter the DCL command RECALL or press CTRL/B. The DCL command RECALL allows you to display a previously entered command. Then, if you want to process the command again, press RETURN. The RECALL command can act as a reminder for previous commands or can save key strokes for long command strings.

The following example shows how to enter a DCL command (SHOW TIME) and then use the RECALL command to display it again:

```
$ SHOW TIME
  10-NOV-1985 13:43:37
$ RECALL
$ SHOW TIME
  10-NOV-1985 13:44:04
```

Add the /ALL qualifier to the RECALL command to display the last twenty commands you have entered. The most recently entered command is number 1. The next to the last command entered is number 2. The RECALL command itself is never assigned a number.

```
$ RECALL/ALL
 1 mail
 2 show time
 3 set default [bellini.opera.costumes]
 4 edit business_matters.dat
 5 show time
 6 mail
 7 set default [bellini.musical.staging]
 8 show users
 9 erase
10 edit ticket_sales.dat
11 type ticket_sales.dat
12 mail
13 show time
14 print ticket_sales.dat
15 show terminal
16 erase
17 edit letter.fun
18 print letter.fun
19 show time
20 mail
```

You can reenter a command by typing its number with the RECALL command, as the following example shows:

```
$ RECALL 14
$ print ticket_sales.dat
```

You can also enter the first character of a command to recall a command. For example, to recall the last command entered that began with the letter "T" (command number 11), enter the following command:

```
$ RECALL T
```

```
$ type ticket_sales.dat
```

If more than one command begins with the letter "T" , you must specify more than one character.

6.4.4 Saving Time by Defining Keys

You can use the DCL command DEFINE/KEY to assign definitions to keypad keys on certain terminals. The terminals include VT52s, VT100 series, and terminals with LK201 keyboards. When you enter the DEFINE/KEY command, you must specify a key-name (such as, PF1) followed by an equivalence-string (such as, the DCL command, DIRECTORY):

```
$ DEFINE/KEY key-name "equivalence-string"
```

For example, you can equate the keypad key PF1 to the DCL command DIRECTORY:

```
$ DEFINE/KEY PF1 "DIRECTORY"
```

You must enclose an equivalence-string in quotation marks (") if the string contains any spaces.

The following example shows how to equate the keypad key PF2 to the DCL command SHOW TIME:

```
$ DEFINE/KEY PF2 "SHOW TIME"
```

To display a key definition, use the DCL command SHOW KEY followed by the name of the key:

```
$ SHOW KEY PF2
DEFAULT keypad definitions:
  PF2 = "show time"
```

To see all the currently defined keys, enter the SHOW KEY command with the /ALL qualifier.

```
$ SHOW KEY/ALL
DEFAULT keypad definitions:
  PF1 = "DIRECTORY"
  PF2 = "SHOW TIME"
```

When you enter the SHOW KEY command specifying an undefined key, DCL displays the following message:

```
%DCL-W-UNDKEY, DEFAULT key PF3 is undefined
```

To undefine keys, use the DELETE/KEY command, as follows:

```
$ DELETE/KEY key-name
```

For example, to undefine the key PF1, enter the following command:

```
$ DELETE/KEY PF1
%DCL-I-DELKEY, DEFAULT key PF1 has been deleted
```

To delete all your key definitions, enter the DCL command DELETE/KEY with the /ALL qualifier.

When you log out, any key definitions you have made during the session are deleted. To maintain key definitions from session to session, list them in your LOGIN.COM file. The following example shows three lines containing key definitions from a LOGIN.COM file:

```
$define/key pf1 "DIRECTORY"
$define/key pf2 "SHOW TIME"
$define/key pf3 "SET DEFAULT [TORTELLINI.LETTERS]"
```

For detailed information about available keys for definition and qualifiers to use with the DEFINE/KEY command, see the *VAX/VMS DCL Dictionary*.

6.5 Working with Physical Devices

A physical device is capable of receiving and storing data. Two examples of physical devices are magnetic tapes and disks, which are mass storage devices. You can perform the following tasks by using the corresponding DCL commands:

- Allocate or deallocate a device (ALLOCATE, DEALLOCATE)
- Make a storage device available or unavailable for processing (MOUNT, DISMOUNT)
- Format a storage device (INITIALIZE)

- Save or restore files from storage devices (BACKUP)
- Set characteristics for:
  - Devices (SET DEVICE)
  - Magnetic tapes (SET MAGTAPE)
  - Line printers (SET PRINTER)
  - Mounted volumes (SET VOLUME)
- Display characteristics for:
  - Devices (SHOW DEVICES)
  - Magnetic tapes (SHOW MAGTAPE)
  - Line printers (SHOW PRINTER)

For information on using these DCL commands to manipulate physical devices, see the *Guide to VAX/VMS Disk and Magnetic Tape Operations.*

6.6 Examining and Controlling Your Environment

Several DCL commands are introduced in this section to aid you in examining and controlling your environment. The *VAX/VMS DCL Dictionary* describes all the available SET and SHOW commands. By adding the specified qualifiers to the various SET and SHOW commands, you can easily change your working environment.

6.6.1 Looking at Your Process

You can use the DCL command SHOW PROCESS to display information about your process.

Figure 6–1 displays the results of the SHOW PROCESS command with all the parts labeled:

Figure 6–1 Using the SHOW PROCESS Command

```
❶ 18-JAN-1984 10:13:40.82     ❷ TTA0:                  ❸ User: FLYNN
❹ Pid: 2100044B               ❺ Proc. name: Macro_writer  ❻ UIC: [DOC1,FLYNN]
❼ Priority:  4                ❽ Default file spec: WORK9:[FLYNN.EXAMPLES]
❾ Devices allocated: TTA0:
```

❶ Date and time the SHOW PROCESS command is issued

❷ Device name of the current SYS$INPUT device (your terminal)

❸ User name

❹ Process identification number, which is a 32-bit binary value that uniquely identifies a process

❺ Process name

❻ User identification code specifying the type of access available to the owner, which is either a pair of alphanumeric or numeric characters

❼ Base execution priority

❽ Default directory

❾ Devices allocated to the process

You can use the DCL command SET PROCESS to change the various characteristics of your process. For example, if you want to change your process name, you enter the SET PROCESS command with the /NAME qualifier, as follows:

```
$ SET PROCESS/NAME = string
```

To set your process name to Macro_Writer, enter the following command:

```
$ SET PROCESS/NAME = "Macro_Writer"
```

To see the newly changed process name, enter the SHOW PROCESS command.

When you log out and back in again, your default process name returns.

To save a process name from session to session, enter the SET PROCESS/NAME command in your LOGIN.COM file, as follows:

```
$SET PROCESS/NAME = "Marvelous_Mabel"
```

The process name Marvelous_Mabel will remain in effect until you change or delete the command line in your LOGIN.COM file.

6.6.2 Looking at Your Terminal Characteristics

This section introduces several qualifiers that you can use with the SET TERMINAL command to modify your terminal characteristics:

/ECHO
/NOECHO
/INSERT
/OVERSTRIKE
/NUMERIC_KEYPAD
/APPLICATION_KEYPAD
/WIDTH
/WRAP
/NOWRAP

To see all your terminal characteristics, enter the DCL command SHOW TERMINAL. You will see a display like the one in Figure 6–2.

The qualifiers discussed in this section affect the characteristics that are highlighted in Figure 6–2.

For detailed information about all the available qualifiers, see the *VAX/VMS DCL Dictionary*.

Figure 6–2 Using the SHOW TERMINAL Command

```
Terminal: _TTA0:       Device_Type: VT100          Owner: Macro_writer(FLYNN)
   Input:   9600      LFfill:  0      Width:  80      Parity: None
   Output:  9600      CRfill:  0      Page:   24
Terminal Characteristics:
   Interactive         Echo               Type_ahead          No Escape
   No Hostsync         TTsync             Lowercase           Tab
   Wrap                Scope              No Remote           No Holdscreen
   No Eightbit         Broadcast          No Readsync         Form
   Fulldup             No Modem           No Local_echo       No Autobaud
   No Hangup           No Brdcstmbx       No DMA              No Altypeahd
   Set_speed           Line Editing       Overstrike editing  No Fallback
   No Dialup           No Secure server   No Disconnect       No Pasthru
   No SIXEL Graphics   No Soft Characters No Printer Port     Numeric Keypad
   ANSI_CRT            No Regis           No Block_mode       Advanced_video
   No Edit_mode        DEC_CRT
```

6.6.2.1 Using the /ECHO and /NOECHO Qualifiers

The qualifiers /ECHO and /NOECHO control whether the terminal displays (echoes) the input lines that it receives. By default, /ECHO is set, allowing you to see what you type. When you use the /NOECHO qualifier, you will not see what you type.

The following example shows how the /ECHO and /NOECHO qualifiers work. The fourth and sixth lines contain what you type, which is invisible. The first undisplayed command you enter is SHOW TIME (fourth line). The second undisplayed command you enter is SET TERMINAL/ECHO (sixth line). Notice that after you enter the SET TERMINAL/ECHO command on the sixth line, the commands you type are displayed again.

```
$ SHOW TIME
17-NOV-1985  15:43:46
$ SET TERMINAL/NOECHO
$
17-NOV-1985  15:46:15
$
$ SHOW TIME
17-NOV-1985  15:43:46
```

6.6.2.2 Using the /INSERT and /OVERSTRIKE Qualifiers

The /INSERT and /OVERSTRIKE qualifiers allow you to either insert characters or type over characters when you are editing command strings. By default, the /OVERSTRIKE qualifier is in effect.

Follow these three steps to see how the /OVERSTRIKE qualifier works:

1. Type the SHOW TIME command, but do not press RETURN.

```
$ SHOW TIME
```

2. Press the left arrow key four times, placing the cursor over the "T" in the word "TIME" .

3. Type the word "USERS" . As you type each character, "USERS" replaces "TIME" .

```
$ SHOW USERS
```

Follow these four steps to see how the /INSERT qualifier works:

1. Enter the SET TERMINAL/INSERT command.

2. Type the SHOW TIME command, but do not press RETURN.

```
$ SHOW TIME
```

3. Press the left arrow key four times, placing the cursor over the "T" in the word "TIME" .

4. Type the word "USERS" . As you type each character, the word "TIME" moves to the right.

```
$ SHOW USERSTIME
```

To reinstate the overstrike characteristic, enter the SET TERMINAL/OVERSTRIKE command.

6.6.2.3 Using the /NUMERIC_KEYPAD and /APPLICATION_KEYPAD Qualifiers

The /NUMERIC_KEYPAD and /APPLICATION_KEYPAD qualifiers allow you to use the keys on the keypad to either type numbers and punctuation marks or to type defined keys. By default, /NUMERIC_KEYPAD is in effect, enabling you to type the numbers and punctuation marks on the keypad. For example, when you press the COMMA key on the keypad, you will see a comma (,).

To take advantage of the /APPLICATION_KEYPAD qualifier, define one of the keypad keys by using the DEFINE/KEY command. For example, enter the following command string:

```
$ DEFINE/KEY COMMA "SHOW TIME"
%DCL-I-DEFDKEY, DEFAULT key COMMA has been defined
```

Then, enter the SET TERMINAL/APPLICATION_KEYPAD command. When you press the COMMA key, you will see the SHOW TIME command instead of a comma.

6.6.2.4 Using the /WIDTH Qualifier

You can use the /WIDTH qualifier with the SET TERMINAL command to specify the number of characters on each input or output line. Use the following syntax:

```
$ SET TERMINAL/WIDTH=n
```

"N" can be any number from 0 through 255. When you specify a number that is greater than 80, your screen displays "thin" characters (on VT100-type and VT200 Series terminals). For example, enter the following command:

```
$ SET TERMINAL/WIDTH=81
```

Immediately, all the characters on your screen become narrow. To retain your previous character size, enter the SET TERMINAL/WIDTH command again specifying a number less than 81.

6.6.2.5 Using the /WRAP and /NOWRAP Qualifiers

The /WRAP and /NOWRAP qualifiers control whether or not the terminal generates a carriage return/line feed when it reaches the end of the line. You can determine the end of a line by setting the terminal width using the /WIDTH qualifier.

To see howha line wraps, press the letter "w" and hold it down. You will see a string of w's on the screen. When the string of w's gets to the end of the line, it goes to the beginning of the next line (or wraps). By default, any string of characters you enter will wrap unless you enter the SET TERMINAL /NOWRAP command.

The first part of the following example displays wrapping characters when the length of the input line is set to 40 and then to 20. Notice the lack of wrap after the SET TERMINAL /NOWRAP command is entered.

```
$ SET TERMINAL/WIDTH=40
$ wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
   .
   .
   .
$ SET TERMINAL/WIDTH=20
$ wwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwww
   .
   .
   .
$ SET TERMINAL/NOWRAP
$ wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
```

6.6.3 For More Information

For more detailed information about the commands presented here, see the *VAX/VMS DCL Dictionary* and the *Guide to VAX/VMS Disk and Magnetic Tape Operations.*

Glossary

account: Enables access to the system software (command interpreters, compilers, utilities, and so on) including the ability to perform work of a general nature (program development, text editing, and so on).

ASCII: American Standard Code for Information Interchange. ASCII is the standard format for sending readable text. It is a code used by many computers to translate letters, numbers, and symbols from a keyboard into machine code, and vice versa.

Thus, an ASCII file is a file that can be read both by people and by computers.

assembler: Language processor that translates a source program containing assembly language directives and machine instructions into an object module.

assembly language: Machine oriented programming language. VAX MACRO is the assembly language for the VAX computer.

assignment statement: Definition of a symbol name to use in place of a character string or numeric value. Symbols can define synonyms for system commands or can be used for variables in command procedures.

batch: Mode of processing in which all commands to be executed by the operating system and, optionally, data to be used as input to the commands are placed in a file or punched onto cards and submitted to the system for execution.

batch job: A noninteractive process.

buffer: A temporary storage area.

command: An instruction or request for the system to perform a particular action. An entire command string consists of the command name with any parameters and/or qualifiers.

command interpreter: The operating system component responsible for reading and translating interactive and batch commands. The default command interpreter for the VAX/VMS operating system is what interprets the DIGITAL Command Language (DCL).

command string: A command with any parameters and/or qualifiers.

command procedure: File containing a sequence of commands to be executed by the operating system. The command procedure can be submitted for execution at the terminal or as a batch job.

compiler: Language processor that translates a source program containing high-level language statements (for example, FORTRAN) into an object module.

concatenate: To link together in a series.

CPU: Central Processing Unit. It is the hardware that handles all calculating and routing of input and output (I/O), as well as executing images. The CPU is the part of the computer that actually computes.

cursor: A flashing indicator used on video terminals to point to the screen position where the next character will appear. It is called a "cursor" because it shows the "course" or direction the printed or typed line will follow.

data: A general term used for any representation of facts, concepts, or instructions in a form suitable for communication, interpretation, or processing. When commands prompt you for command elements, they are asking you for data to process.

DCL: DIGITAL Command Language. It provides a means of communication between the user and the operating system. DCL is designed for ease of use. Commands are English words, and if necessary elements are not typed in, DCL will prompt you for them.

debugger: Interactive program that allows you to display and modify program variables during execution and to step through a program to locate and detect programming errors.

default: Value supplied by the system when a user does not specify a required command parameter or qualifier.

default disk: The disk from which the system reads and to which the system writes, by default, all files that you create. The default is used whenever a file specification in a command does not explicitly name a device.

delimiter: A character that separates, terminates, or organizes elements of a character string or statement. For example, in the file specification, STORIES.DAT, the period (.) is the delimiter that enables the system to tell the difference between the file name STORIES and the file type DAT.

device: Any peripheral hardware connected to the processor and capable of receiving, storing, or transmitting data. Line printers and terminals are examples of record-oriented devices. Magnetic tapes and disks are examples of mass-storage devices. Terminal line interfaces and interprocessor links are examples of communications devices. All devices have names either in the form ddnn:, where dd is a two-letter mnemonic, nn is an octal number, and the colon(:) is a required terminator or as a logical name.

device name: Identification of a physical device (for example, DBA2) or a logical name (for example, SYS$OUTPUT) that is equated to a physical device name.

directory: A file that briefly catalogs a set of files stored on disk or tape. The directory includes the name, type, and version number of each file in the set.

disk: High-speed, random-access devices. There are several kinds of disks. Floppy disks are small, flexible disks. Hard disks are either fixed in place or removable. Removable disk types include a single hard disk enclosed in a protective case and a stacked set of disks enclosed in a protective case.

echo: The display of a character either on the screen or hard copy that was typed on a terminal keyboard. Terminals are dual devices, sending input and receiving output. Typing on the terminal is sending input to the computer. Echoing is receiving output from the computer.

editor: Program that creates or modifies files. In VAX/VMS, the default system editor is EDT, which is interactive.

equivalence name: Character string equated to a logical name. When a command or program refers to a file or device by its logical name, the system translates the logical name to its predefined equivalence name.

error message: Sent by the system when some action you have requested fails. Each error message identifies the particular part that detected the error. The great majority of error messages result from typing mistakes or mistakes in syntax. Often, you can correct the error by retyping the command.

field: Usually refers to a portion of a command or a command element. For example, the file name and file type are two fields of a file specification.

file: Collection of data treated as a unit; generally used to refer to data stored on magnetic tapes or disks.

file name: The name component of a file specification.

file specification: Unique identification of a file. A file specification describes the physical location of the file, as well as file name and file type identifiers that describe the file and its contents.

file type: The type component of a file specification. A file type generally describes the nature of a file, or how it is used. For example, FOR indicates a FORTRAN source program.

folder: A subdivision of a file in which you can store mail messages.

form feed: Analogous to a line feed, but instead of moving down one line to resume printing, the line printer moves past the perforations in the paper to the top of a new form or page. A form feed consists of a number of line feeds.

functionality: A computer industry term for what the hardware or software can do.

global symbol: A symbol defined with an assignment statement that is recognized in any command procedure that is executed.

hanging: A terminal or process that appears to be going nowhere or doing nothing. Hung terminals are sometimes described as static, dormant, or locked. Hung terminals may result from a busy system, a crash, or unavailability of system resources.

hard-copy terminal: Terminals that print output on paper are called hard-copy terminals.

hardware: The physical computer equipment, including such mechanical devices as the line printer, the terminals, the mass-storage devices, and so forth.

header page: Printed page at the beginning of a listing that identifies the printed file.

help file: A text file in a format suitable for use with the HELP command. Help files can include simply organized information and can provide up to nine levels of search.

high-level language: Transportable programming language, such as BASIC, FORTRAN, or COBOL. Programs in these languages are not tied to a particular kind of computer. Each programming statement in a high-level language is translated into several machine-language instructions.

image: Output from the linker, created from processing one or more object modules. An image is the executable version of a program.

input file: File containing data to be transferred into the computer.

> One common mistake made in using the system is confusing input and output files. DCL usually prompts for these files, but most system utilities require you to identify your input and output files by position in a command line. You should be sure of the syntax for the command you are using.

interactive: Mode of communication with the operating system in which a user enters a command, and the system executes it and responds.

job: (1) The accounting unit equivalent to a process; jobs are classified as batch or interactive. (2) A print job.

K: A unit for measuring the size of memory or similar resources. K is short for kilo and is used roughly to mean 1000, although formally K is equal to 1024.

keypad: The small set of keys next to the main keyboard on a terminal.

keyword: A command name, qualifier, or option. Keywords must be typed verbatim or truncated according to the rules of DCL.

lexical function: A command language construct that the command interprer evaluates and substitutes before it parses a command string. Lexical functions return information about the current process (for example, the UIC or default directory) and about character strings (for example, their length or the location of substrings).

line editor: Program that allows you to make additions and deletions to a file on a line by line basis.

line printer: An output device that prints files a line at a time. It is used for printing large amounts of output that would otherwise tie up a slower device.

linker: Program that creates an executable program, called an image, from one or more object modules produced by a language compiler or assembler. Programs must be linked before they can be executed.

local symbol: A symbol defined with an assignment statement that is recognized only within the command procedure in which it is defined.

log in: To perform a sequence of actions at a terminal that establishes a user's communication with the operating system and sets up default characteristics for the user's terminal session.

log out: To terminate interactive communication with the operating system. The LOGOUT command executes the procedure an d ends a terminal session.

logical name: Character string used to refer to files or devices by other than their specific names. A command or program can refer to a file by a logical name; the logical name can be equated to an equivalence name at any time; when the command or program refers to the logical name, the system translates the logical name to its defined equivalence name.

logical name table: A table that contains a set of logical names and their equivalence names for a particular process, a particular group, or the system.

machine code: A sequence of binary machine instructions in a form executable by the computer.

magnetic tape: Medium on which data can be stored and accessed.

mass-storage device: An input/output device where data and other types of files are stored while they are not being used. Typical mass-storage devices include disks, magnetic tapes, floppy disks, and DECtapes. Each mass-storage device uses a particular magnetic medium to hold its data.

memory: A series of physical locations into which data or instructions can be placed in the form of binary words. Each location in memory can be addressed and its contents can be altered.

network: A collection of interconnected computer systems.

node: An individual computer system in a network that can communicate with other computer systems in the network.

node specification: The component of a file specification that identifies the location of a computer system in a network of computer systems.

object module: Output from a language compiler or assembler that can be linked with other object modules to produce an executable image. An object module is a file with a file type of OBJ.

operating system: The system software that controls the operations of the computer.

operator: A person responsible for maintaining the system. Within small systems, the job may be combined with that of the system manager or informally divided among several people. The responsibilities of the operator include changing ribbons, rebooting the system, and keeping records.

output file: File to which the computer transfers data.

parameter: Object of a command. A parameter can be a file specification, a symbol value passed to a command procedure, or a word defined by the DIGITAL Command Language.

parse: Break a command string into its elements to interpret it. For example, a PRINT command without a file specification, or with illegal characters in the file specification, will not parse correctly.

password: Protective word associated with a user name. A user logging in to the system must supply the correct password before the system will permit access.

peripheral devices: Any unit, distinct from the CPU and memory, that can provide the system with input or accept output from it. Terminals, line printers, and disks are peripheral devices.

priority: A rank assigned to a process to determine its precedence in obtaining system resources when the process is running.

program: A series of instructions written for the computer to follow.

prompt: A symbol used by the system as a cue to signal that the system is ready to accept input from you.

protection code: Specifies what access different categories of system users may have to the file and what they may do to the file when they access it.

qualifier: Command modifier that describes the operation of a command. A qualifier is always preceded by a slash character (/).

queue: A line of items waiting to be processed.

random access: This term refers to memory or mass-storage devices where all information is equally accessible. With random access, the next location from which data is to be obtained is not dependent on the location of the last data obtained. All records appear to be adjacent on a random-access device. As far as the use is concerned, there is no beginning, middle, or end to the data.

range specification: Used with the EDT line editor to define the line(s) to be affected by the editing command.

reverse video: A feature of the VT100 terminal that reverses the default video contrast. If black figures on a white background is the default, reverse video displays white on black. Reverse video is used with some EDT keypad commands to highlight a range of text.

RMS: Record Management Services. RMS is a sophisticated set of routines used to open and close files, read from files, and extend and delete files.

scrolling: When more than a screenful of output is sent to a video terminal, the output scrolls up. New output appears at the bottom of the screen and eventually disappears when it reaches the top, just as if it were on a scroll that is being unrolled at the bottom and rolled at the top.

sequential access: Records or files read one after another in the order in which they appear in the file or volume. Magnetic tape is a sequential-access medium. If you are half way through the tape and wish to read some record that is a third of the way throught the tape, you must go back to the beginning and read through until you get to the record that you want.

software: The collection of images, procedures, rules, and documentation associated with the operation of a particular computer system. For example, the operating system is software.

source program: A program written in text form that must be compiled or assembled to be used.

string: A sequence of characters. When you use an editor to search for a word or phrase, you are searching for a string. The sequence of characters that forms a command is often called a command string.

subdirectory: Directory file cataloged in a higher-level directory that lists additional files belonging to the owner of the directory.

subroutine: A routine that can be used as part of another routine. For instance, you might write a routine to print the time in large numbers on your terminal. You could then call that routine as a subroutine in some task that required printing the time in large numbers.

switch hook character: The Phone Utility prompt. You must type the switch hook character (which is a percent sign by default) before you enter a PHONE command.

symbol: An entity that must be defined, or given a meaning, so that it can be used.

syntax: The form that a command must follow. Misspelled words are the most common syntax errors.

system manager: Person who makes resources available to users and sets up restrictions governing the use of such resources.

terminal: Hardware communication device, with a typewriter-like keyboard that receives and transmits information between users and the system.

timesharing: A time-sharing system is a system in which each user gets equal computer time in turn. This is in contrast to the allocation based on need and priority in a real-time system.

UIC: User identification code. This code identifies a user by a group number and a member number. (Both numbers are enclosed in brackets.)

user name: Name by which the system identifies a particular user. To gain access to the system, a user specifies a user name followed by a password.

utility: A general-purpose program that performs tasks included in an operating system to perform common functions, such as editing or file handling.

version number: Numeric component of a file specification. When a file is edited, its version number is increased by one.

volume: The largest logical unit of the file structure. A volume contains files and corresponds to a physical unit of mass storage.

wildcard character: A symbol used with many DCL commands in place of all or part of a file specification to refer to several files rather than specifying them individually.

Index

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

Cut Along Dotted Line